aws

Guía para desarrolladores, versión 1

AWS IoT Greengrass

AWS IoT Greengrass: Guía para desarrolladores, versión 1

Table of Contents

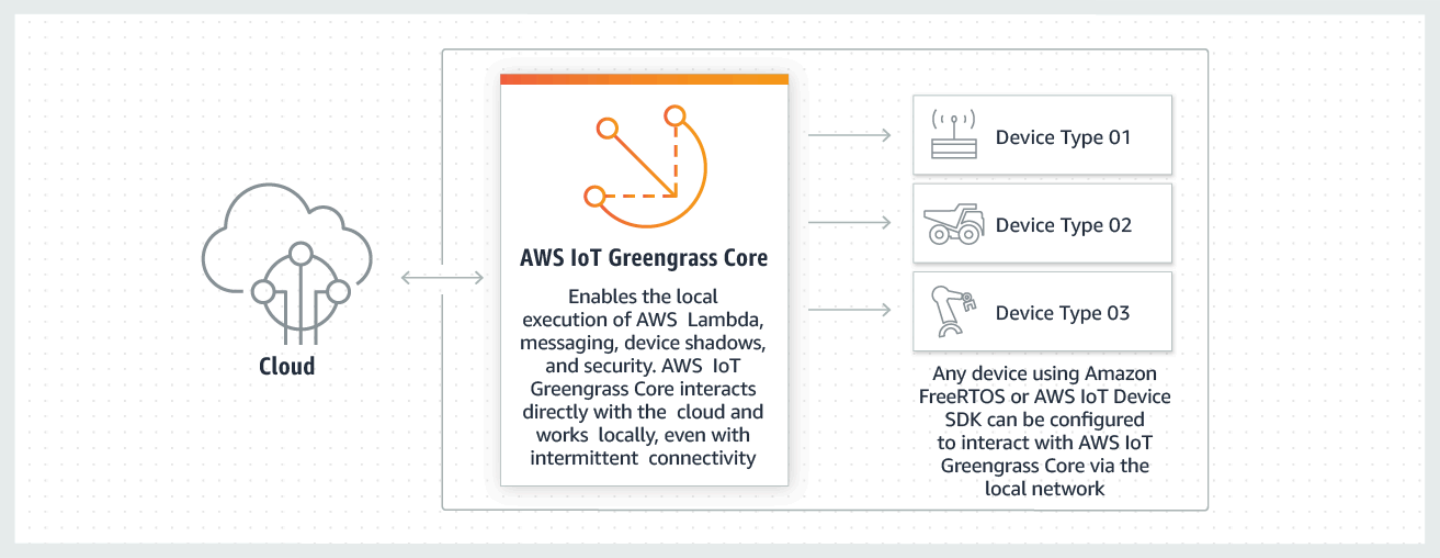
AWS IoT Greengrass Version 1 entró en la fase de vida útil prolongada el 30 de junio de 2023. Para obtener más información, consulte la política de mantenimiento de AWS IoT Greengrass V1 Después de esta fecha, AWS IoT Greengrass V1 no se publicarán actualizaciones que proporcionen funciones, mejoras, correcciones de errores o parches de seguridad. Los dispositivos que se ejecuten AWS IoT Greengrass V1 no se verán afectados y seguirán funcionando y conectándose a la nube. Le recomendamos encarecidamente que migre a AWS IoT Greengrass Version 2, ya que añade importantes funciones nuevas y es compatible con plataformas adicionales.

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.

¿Qué es AWS IoT Greengrass?

AWS IoT Greengrass es un software que extiende las capacidades de la nube a los dispositivos locales. Esto permite que los dispositivos recopilen y analicen datos más cerca del origen de la información, reaccionen de forma autónoma a eventos locales y se comuniquen de forma segura entre sí en las redes locales. Los dispositivos locales también pueden comunicarse de forma segura con los datos de IoT AWS IoT Core y exportarlos a Nube de AWS. AWS IoT Greengrass los desarrolladores pueden usar AWS Lambda funciones y conectores prediseñados para crear aplicaciones sin servidor que se implementen en los dispositivos para su ejecución local.

En el siguiente diagrama se muestra la arquitectura básica de. AWS IoT Greengrass



AWS IoT Greengrass permite a los clientes crear dispositivos de IoT y lógica de aplicaciones. En concreto, AWS IoT Greengrass proporciona una gestión basada en la nube de la lógica de las aplicaciones que se ejecuta en los dispositivos. Los conectores y las funciones de Lambda implementados localmente se activan por eventos locales, mensajes de la nube u otras fuentes.

En AWS IoT Greengrass, los dispositivos se comunican de forma segura en una red local e intercambian mensajes entre sí sin tener que conectarse a la nube. AWS IoT Greengrass proporciona un administrador de mensajes pub/sub local que puede almacenar en búfer los mensajes de forma inteligente si se pierde la conectividad, de modo que se conservan los mensajes entrantes y salientes a la nube.

AWS IoT Greengrass protege los datos de los usuarios:

- A través de la autorización y autenticación seguras de los dispositivos.

- A través de conectividad segura en la red local.
- Entre los dispositivos locales y la nube.

Las credenciales de seguridad de los dispositivos funcionan dentro de un grupo hasta que se revocan, aunque se interrumpa la conectividad a la nube, de manera que los dispositivos puedan seguir comunicándose de forma segura en el nivel local.

AWS IoT Greengrass proporciona over-the-air actualizaciones seguras de las funciones de Lambda.

AWS IoT Greengrass consta de:

- Distribuciones de software
  - AWS IoT Greengrass Software básico
  - AWS IoT Greengrass SDK principal
- Servicio en la nube
  - AWS IoT Greengrass API
- Características
  - Tiempo de ejecución de Lambda
  - Implementación de sombras
  - Administrador de mensajes
  - Administración de grupos
  - Servicio de detección
  - O agente ver-the-air de actualización
  - Administrador de transmisiones
  - Acceso a recursos locales
  - Inferencia de aprendizaje automático local
  - Secrets Manager local
  - Conectores con integración incorporada con servicios, protocolos y software

Temas

- SDK
- Plataformas compatibles y requisitos
- AWS IoT Greengrass descargas
- Esperamos tener noticias suyas
- Instalación del software AWS IoT Greengrass Core
- Configuración de AWS IoT Greengrass Core

AWS IoT Greengrass Software básico

El software AWS IoT Greengrass Core ofrece las siguientes funciones:

- Implementación y ejecución local de conectores y funciones de Lambda.
- Procese los flujos de datos localmente con exportaciones automáticas al Nube de AWS.
- Mensajes MQTT a través de la red local entre dispositivos, conectores y funciones de Lambda mediante suscripciones administradas.
- La mensajería MQTT entre AWS IoT dispositivos, conectores y funciones Lambda mediante suscripciones gestionadas.
- Proteja las conexiones entre los dispositivos y la autenticación y autorización de los dispositivos que los Nube de AWS utilizan.
- Sincronización de sombras locales de dispositivos. Las sombras se pueden configurar para sincronizarse con Nube de AWS.
- Acceso controlado a los recursos del dispositivo local y el volumen.
- Implementación de modelos de machine learning entrenados en la nube para la ejecución de la inferencia local.
- Detección automática de direcciones IP que permite a los dispositivos detectar el dispositivo principal de Greengrass.
- Implementación central de la configuración de grupos nuevos o actualizados. Una vez descargados los datos de configuración, el dispositivo principal se reinicia automáticamente.
- Actualizaciones de software seguras over-the-air (OTA) de funciones Lambda definidas por el usuario.
- Almacenamiento seguro y cifrado de los secretos locales y acceso controlado por conectores y funciones de Lambda.

AWS IoT Greengrass las instancias principales se configuran mediante AWS IoT Greengrass API que crean y actualizan las definiciones de AWS IoT Greengrass grupos almacenadas en la nube.

AWS IoT Greengrass Versiones principales del software

AWS IoT Greengrass proporciona varias opciones para instalar el software AWS IoT Greengrass principal, incluidos los archivos de descarga tar.gz, un script de inicio rápido e `apt` instalaciones en plataformas Debian compatibles. Para obtener más información, consulte the section called “Instalación del software AWS IoT Greengrass Core”.

En las siguientes pestañas se describen las novedades y los cambios en las versiones AWS IoT Greengrass del software Core.

GGC v1.11

1.11.6

Mejoras y correcciones de errores:

- Resiliencia mejorada en caso de que se produzca una pérdida repentina de energía durante una implementación.
- Se solucionó un problema por el que la corrupción de los datos del administrador de transmisiones podía impedir que se AWS IoT Greengrass iniciara el software Core.
- Se ha corregido un problema que provocaba que los nuevos dispositivos cliente no se pudieran conectar al núcleo en determinadas situaciones.
- Se ha corregido un error que provocaba que los nombres de los flujos del administrador de flujos no pudieran contener `.log`.

1.11.5

Mejoras y correcciones de errores:

- Mejoras de rendimiento generales y correcciones de errores.

1.11.4

Mejoras y correcciones de errores:

- Se ha corregido un problema con el administrador de transmisiones que impedía actualizar el software AWS IoT Greengrass Core a la versión 1.11.3. Si utilizas Stream Manager para exportar datos a la nube, ahora puedes usar una actualización OTA para actualizar una versión anterior, v1.x, del software Core a la AWS IoT Greengrass versión 1.11.4.
- Mejoras de rendimiento generales y correcciones de errores.

1.11.3

Mejoras y correcciones de errores:

- Se ha corregido un problema que provocaba que el software AWS IoT Greengrass Core que se ejecutaba en un abrir y cerrar de ojos en un dispositivo Ubuntu dejara de responder tras un repentino corte de energía en el dispositivo.
- Se ha corregido un problema que provocaba un retraso en la entrega de los mensajes MQTT a funciones de Lambda de larga duración.
- Se ha corregido un problema que provocaba que los mensajes MQTT no se enviaran correctamente cuando el valor `maxWorkItemCount` estaba establecido en un valor superior a `1024`.
- Se ha corregido un problema que provocaba que el agente de actualización de OTA ignorara el período `KeepAlive` de MQTT especificado en la propiedad `keepAlive` en `config.json`.
- Mejoras de rendimiento generales y correcciones de errores.

> **Important**
>
> Si utilizas el administrador de transmisiones para exportar datos a la nube, no actualices el software AWS IoT Greengrass Core v1.11.3 desde una versión anterior, v1.x. Si es la primera vez que habilitas Stream Manager, te recomendamos encarecidamente que instales primero la última versión del software Core. AWS IoT Greengrass

1.11.1

Mejoras y correcciones de errores:

- Se ha corregido un problema que provocaba que el administrador de flujos aumentara el uso de memoria.
- Se ha corregido un problema que provocaba que el administrador de transmisiones restableciera el número de secuencia de la transmisión `0` si el dispositivo principal de Greengrass estaba apagado durante más tiempo del período especificado time-to-live (TTL) de los datos de la transmisión.
- Se ha corregido un problema que impedía que el administrador de flujos detuviera correctamente los reintentos de exportación de datos al Nube de AWS.

1.11.0

Nuevas características:

- Un agente de telemetría del núcleo de Greengrass recopila datos de telemetría locales y los publica en. Nube de AWS Para recuperar los datos de telemetría para su posterior procesamiento, los clientes pueden crear una EventBridge regla de Amazon y suscribirse a un objetivo. Para obtener más información, consulte Recopilación de datos de telemetría del estado del sistema desde los dispositivos principales. AWS IoT Greengrass
- Una API HTTP local devuelve una instantánea del estado actual de los procesos de trabajo locales iniciados por. AWS IoT Greengrass Para obtener más información, consulte Llamadas a la API de comprobación de estado local.
- Un administrador de transmisiones exporta automáticamente los datos a Amazon S3 y AWS IoT SiteWise.

  Los nuevos parámetros del administrador de flujos le permiten actualizar las transmisiones existentes y pausar o reanudar la exportación de datos.
- Compatibilidad para ejecutar funciones Lambda de Python 3.8.x en el dispositivo principal.
- Una nueva propiedad `ggDaemonPort` en `config.json` que se usa para configurar el número de puerto IPC de Greengrass core. El número de puerto predeterminado es 8000.

  Una propiedad `systemComponentAuthTimeout` nueva en `config.json` que se utiliza para configurar el tiempo de espera de la autenticación IPC de Greengrass core. El valor de tiempo predeterminado es 5000 milisegundos.
- Se aumentó la cantidad máxima de AWS IoT dispositivos por AWS IoT Greengrass grupo de 200 a 2500.

  Se aumentó el número máximo de suscripciones por grupo de 1000 a 10 000.

  Para obtener más información, consulte Puntos de conexión y cuotas de AWS IoT Greengrass.

Mejoras y correcciones de errores:

- Optimización general que puede reducir la utilización de memoria de los procesos de servicio de Greengrass.
- Un nuevo parámetro de configuración del tiempo de ejecución (`mountAllBlockDevices`) permite a Greengrass utilizar los montajes de enlace para montar todos los dispositivos de bloques en un contenedor después de configurar el OverlayFS. Esta característica resolvió

un problema que provocaba un error en la implementación de Greengrass si `/usr` no estaba por debajo de la jerarquía `/`.

- Se ha corregido un problema que provocaba un fallo en el AWS IoT Greengrass núcleo si `/tmp` se trataba de un enlace simbólico.
- Se ha corregido un error para permitir que el agente de implementación de Greengrass elimine de la carpeta los artefactos del modelo de machine learning no utilizados. `mlmodel_public`
- Mejoras de rendimiento generales y correcciones de errores.

Extended life versions

1.10.5

Mejoras y correcciones de errores:

- Mejoras de rendimiento generales y correcciones de errores.

1.10.4

Mejoras y correcciones de errores:

- Se ha corregido un problema que provocaba que el software AWS IoT Greengrass Core que se ejecutaba en un abrir y cerrar de ojos en un dispositivo Ubuntu dejara de responder tras un corte repentino de energía en el dispositivo.
- Se ha corregido un problema que provocaba un retraso en la entrega de los mensajes MQTT a funciones de Lambda de larga duración.
- Se ha corregido un problema que provocaba que los mensajes MQTT no se enviaran correctamente cuando el valor `maxWorkItemCount` estaba establecido en un valor superior a `1024`.
- Se ha corregido un problema que provocaba que el agente de actualización de OTA ignorara el período `KeepAlive` de MQTT especificado en la propiedad `keepAlive` en `config.json`.
- Mejoras de rendimiento generales y correcciones de errores.

1.10.3

Mejoras y correcciones de errores:

- Una propiedad `systemComponentAuthTimeout` nueva en `config.json` que se utiliza para configurar el tiempo de espera de la autenticación IPC de Greengrass core. El valor de tiempo predeterminado es 5000 milisegundos.
- Se ha corregido un problema que provocaba que el administrador de flujos aumentara el uso de memoria.

1.10.2

Mejoras y correcciones de errores:

- Una nueva `mqttOperationTimeout` propiedad en config.json que se utiliza para establecer el tiempo de espera de las operaciones de publicación, suscripción y cancelación de la suscripción en las conexiones de MQTT. AWS IoT Core
- Mejoras de rendimiento generales y correcciones de errores.

1.10.1

Mejoras y correcciones de errores:

- El administrador de secuencias es más resistente a la corrupción de datos de archivos.
- Se ha corregido un problema que provocaba un error de montaje de sysfs en dispositivos que utilizaban el kernel de Linux 5.1 y versiones posteriores.
- Mejoras de rendimiento generales y correcciones de errores.

1.10.0

Nuevas características:

- Un administrador de flujos que procesa flujos de datos localmente y los exporta automáticamente a la nube de Nube de AWS . Esta característica requiere Java 8 en el dispositivo central de Greengrass. Para obtener más información, consulte Administrar secuencias de datos.
- Un nuevo conector de implementación de aplicación de Docker de Greengrass que ejecuta una aplicación Docker en un dispositivo principal. Para obtener más información, consulte the section called “Implementación de aplicaciones Docker”.
- Un nuevo SiteWise conector de IoT que envía datos de dispositivos industriales desde los servidores OPC-UA a las propiedades de los activos. AWS IoT SiteWise Para obtener más información, consulte the section called “IoT SiteWise”.
- Las funciones de Lambda que se ejecutan sin creación de contenedores pueden acceder a los recursos de machine learning del grupo Greengrass. Para obtener más información, consulte the section called “Acceso a recursos de aprendizaje automático”.

- Support para sesiones persistentes de MQTT con AWS IoT. Para obtener más información, consulte the section called “Sesiones persistentes de MQTT con AWS IoT Core”.
- El tráfico MQTT local puede desplazarse a través de un puerto distinto del puerto predeterminado 8883. Para obtener más información, consulte the section called “Puerto MQTT para la mensajería local”.
- Nuevas opciones de `queueFullPolicy` en el SDK de AWS IoT Greengrass Core para la publicación de mensajes de confianza desde funciones de Lambda.
- Compatibilidad para ejecutar funciones Lambda de Node.js 12.x en el dispositivo principal.
- Las actualizaciones O ver-the-air (OTA) con integración de seguridad de hardware se pueden configurar con OpenSSL 1.1.
- Mejoras de rendimiento generales y correcciones de errores.

1.9.4

Mejoras y correcciones de errores:

- Mejoras de rendimiento generales y correcciones de errores.

1.9.3

Nuevas características:

- Support para ARMv6L. AWS IoT Greengrass El software principal v1.9.3 o posterior se puede instalar en distribuciones de Raspbian en arquitecturas ARMv6L (por ejemplo, en dispositivos Raspberry Pi Zero).
- Actualizaciones OTA en el puerto 443 con ALPN. Los núcleos Greengrass que utilizan el puerto 443 para el tráfico MQTT ahora admiten actualizaciones de software over-the-air (OTA). AWS IoT Greengrass utiliza la extensión TLS de Application Layer Protocol Network (ALPN) para habilitar estas conexiones. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core y the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.

Mejoras y correcciones de errores:

- Se ha corregido un error introducido en la versión 1.9.0 que impedía que las funciones de Lambda de Python 2.7 enviaran cargas binarias a otras funciones de Lambda.
- Mejoras de rendimiento generales y correcciones de errores.

1.9.2

Nuevas características:

- Support for OpenWrt. AWS IoT Greengrass El software principal v1.9.2 o posterior se puede instalar en OpenWrt distribuciones con arquitecturas Armv8 (AArch64) y ARMv7L. OpenWrt Actualmente, no admite la inferencia de aprendizaje automático.

1.9.1

Mejoras y correcciones de errores:

- Corrige un error introducido en la versión 1.9.0 que elimina los mensajes de `cloud` que contienen caracteres comodín en el tema.

1.9.0

Nuevas características:

- Compatibilidad con los tiempos de ejecución Lambda de Node.js 8.10 y de Python 3.7. Las funciones Lambda que utilizan los tiempos de ejecución de Python 3.7 y Node.js 8.10 ahora pueden ejecutarse en un núcleo. AWS IoT Greengrass (AWS IoT Greengrass sigue siendo compatible con los tiempos de ejecución de Python 2.7 y Node.js 6.10).
- Conexiones MQTT optimizadas. El núcleo de Greengrass establece menos conexiones con AWS IoT Core. Este cambio puede reducir los costos operativos de los cargos que se basan en el número de conexiones.
- Clave Curva elíptica (EC) para el servidor MQTT local. El servidor MQTT local admite claves EC además de las claves RSA. (El certificado del servidor MQTT tiene una firma SHA-256 RSA, independientemente del tipo de clave). Para obtener más información, consulte the section called “Entidades de seguridad”.

Mejoras y correcciones de errores:

- Mejoras de rendimiento generales y correcciones de errores.

1.8.4

Se ha corregido un problema con la sincronización de sombras y la reconexión del administrador de certificados de dispositivo.

Mejoras de rendimiento generales y correcciones de errores.

1.8.3

Mejoras de rendimiento generales y correcciones de errores.

1.8.2

Mejoras de rendimiento generales y correcciones de errores.

1.8.1

Mejoras de rendimiento generales y correcciones de errores.

1.8.0

Nuevas características:

- Identidad de acceso predeterminada configurable para las funciones de Lambda del grupo. Esta configuración de nivel de grupo determina los permisos predeterminados que se utilizan para ejecutar funciones de Lambda. Puede definir el ID de usuario, el ID de grupo o ambos. Las funciones de Lambda pueden invalidar la identidad de acceso predeterminada de su grupo. Para obtener más información, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.
- Tráfico HTTPS a través del puerto 443. La comunicación HTTPS se puede configurar para la comunicación a través del puerto 443 en lugar del puerto 8443 predeterminado. Esto complementa la AWS IoT Greengrass compatibilidad con la extensión TLS de Application Layer Protocol Network (ALPN) y permite que todo el tráfico de mensajería de Greengrass, tanto MQTT como HTTPS, utilice el puerto 443. Para obtener más información, consulte the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.
- Identificadores AWS IoT de cliente con nombres predecibles para las conexiones. Este cambio proporciona compatibilidad con los AWS IoT Device Defender y eventos de ciclo de vida de AWS IoT, para que puede recibir notificaciones de eventos de conexión, desconexión, suscripción y anulación de suscripción. Los nombres predecibles también permiten crear lógica en torno a los ID de conexión (por ejemplo, crear plantillas de política de suscripción basadas en atributos del certificado). Para obtener más información, consulte the section called “ID de cliente para conexiones MQTT con AWS IoT”.

Mejoras y correcciones de errores:

- Se ha corregido un problema con la sincronización de sombras y la reconexión del administrador de certificados de dispositivo.
- Mejoras de rendimiento generales y correcciones de errores.

1.7.1

Nuevas características:

- Los conectores Greengrass proporcionan una integración integrada con la infraestructura local, AWS los protocolos de los dispositivos y otros servicios en la nube. Para obtener más información, consulte Integración con servicios y protocolos mediante conectores.

- AWS IoT Greengrass se extiende AWS Secrets Manager a los dispositivos principales, lo que hace que sus contraseñas, tokens y otros secretos estén disponibles para los conectores y las funciones Lambda. Los secretos se cifran en reposo y en tránsito. Para obtener más información, consulte Implementación de secretos en el núcleo.
- Compatibilidad con una raíz de hardware de opción de seguridad de confianza. Para obtener más información, consulte the section called “Integración de la seguridad de hardware”.
- La configuración de aislamiento y permiso que permite que las funciones de Lambda se ejecuten sin contenedores de Greengrass y usen los permisos de un usuario y un grupo especificados. Para obtener más información, consulte the section called “Control de la ejecución de la función de Lambda de Greengrass”.
- Puede ejecutar AWS IoT Greengrass en un contenedor Docker (en Windows, macOS o Linux) configurando su grupo de Greengrass para que se ejecute sin contenerización. Para obtener más información, consulte the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”.
- Mensajería MQTT en el puerto 443 con negociación de protocolo de capa de aplicaciones (ALPN) o conexión a través de un proxy de red. Para obtener más información, consulte the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.
- El motor de ejecución de aprendizaje profundo de SageMaker Neo, que admite modelos de aprendizaje automático optimizados por el compilador de aprendizaje profundo de SageMaker Neo. Para obtener información sobre el entorno de ejecución de aprendizaje profundo de Neo, consulte the section called “Entornos de ejecución y bibliotecas para la inferencia de machine learning”.
- Compatibilidad con Raspbian Stretch (2018-06-27) en dispositivos del núcleo de Raspberry Pi.

Mejoras y correcciones de errores:

- Mejoras de rendimiento generales y correcciones de errores.

Además, las siguientes características están disponibles con esta versión:

- El comprobador de AWS IoT dispositivos AWS IoT Greengrass, que puede utilizar para comprobar que funcionan la arquitectura de la CPU, la configuración del núcleo y los controladores. AWS IoT Greengrass Para obtener más información, consulte Uso AWS IoT del comprobador de dispositivos para V1 AWS IoT Greengrass.
- Los paquetes de software AWS IoT Greengrass Core, AWS IoT Greengrass Core SDK y AWS IoT Greengrass Machine Learning SDK están disponibles para su descarga a través

de Amazon CloudFront. Para obtener más información, consulte the section called “AWS IoT Greengrass descargas”.

1.6.1

Nuevas características:

- Los ejecutables de Lambda que ejecutan un código binario en el núcleo de Greengrass. Utilice el nuevo AWS IoT Greengrass Core SDK para C para escribir ejecutables Lambda en C y C++. Para obtener más información, consulte the section called “Ejecutables de Lambda”.
- Opción de almacenamiento en caché local de mensajes que pueden persistir entre reinicios. Puede definir la configuración de almacenamiento para mensajes MQTT que se ponen a la cola para su procesamiento. Para obtener más información, consulte the section called “Cola de mensajes MQTT”.
- Intervalo configurable de reintentos de reconexión máximos para cuando el dispositivo del núcleo está desconectado. Para obtener más información, consulte la propiedad `mqttMaxConnectionRetryInterval` en the section called “Archivo de configuración de AWS IoT Greengrass Core”.
- Acceso del recurso local al directorio /proc del host. Para obtener más información, consulte Acceder a recursos locales.
- Directorio de escritura configurable. El software AWS IoT Greengrass Core se puede implementar en ubicaciones de solo lectura y lectura-escritura. Para obtener más información, consulte the section called “Configurar un directorio de escritura”.

Mejoras y correcciones de errores:

- Se ha mejorado el rendimiento de la publicación de mensajes en el núcleo de Greengrass y entre los dispositivos y el núcleo.
- Se han reducido los recursos de computación necesarios para procesar los registros generados por funciones de Lambda definidas por el usuario.

1.5.0

Nuevas características:

- AWS IoT Greengrass La inferencia de Machine Learning (ML) está disponible de forma general. Puede llevar a cabo inferencias de aprendizaje automático localmente en los dispositivos de AWS IoT Greengrass con modelos creados y entrenados en la nube. Para obtener más información, consulte Cómo realizar la inferencia de machine learning.

- Las funciones de Lambda de Greengrass ahora admiten datos binarios como carga de entrada, además de JSON. Para usar esta función, debes actualizar a la versión 1.1.0 del SDK de AWS IoT Greengrass Core, que puedes descargar desde la página de descargas del SDK de AWS IoT Greengrass Core.

Mejoras y correcciones de errores:

- Se ha reducido la huella de memoria global.
- Se han realizado mejoras de desempeño para enviar mensajes a la nube.
- Se han realizado mejoras de desempeño y de estabilidad en el agente de descargas, el administrador de certificados de dispositivos y el agente de actualización OTA.
- Correcciones de errores menores.

1.3.0

Nuevas características:

- O ver-the-air (OTA) agente de actualización capaz de gestionar trabajos de actualización de Greengrass desplegados en la nube. El agente se encuentra en el nuevo directorio `/greengrass/ota`. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.
- La característica de acceso a recursos locales permite a las funciones de Lambda de Greengrass obtener acceso a los recursos locales, como dispositivos periféricos y volúmenes. Para obtener más información, consulte Acceder a recursos locales con conectores y funciones de Lambda.

1.1.0

Nuevas características:

- AWS IoT Greengrass Los grupos implementados se pueden restablecer eliminando las funciones, suscripciones y configuraciones de Lambda. Para obtener más información, consulte the section called “Restablecimiento de implementaciones”.
- Compatibilidad con los tiempos de ejecución de Lambda de Node.js 6.10 y Java 8, además de Python 2.7.

Para migrar desde la versión anterior del AWS IoT Greengrass núcleo:

- Copie los certificados de la carpeta `/greengrass/configuration/certs` en `/greengrass/certs`.
- Copie `/greengrass/configuration/config.json` en `/greengrass/config/config.json`.

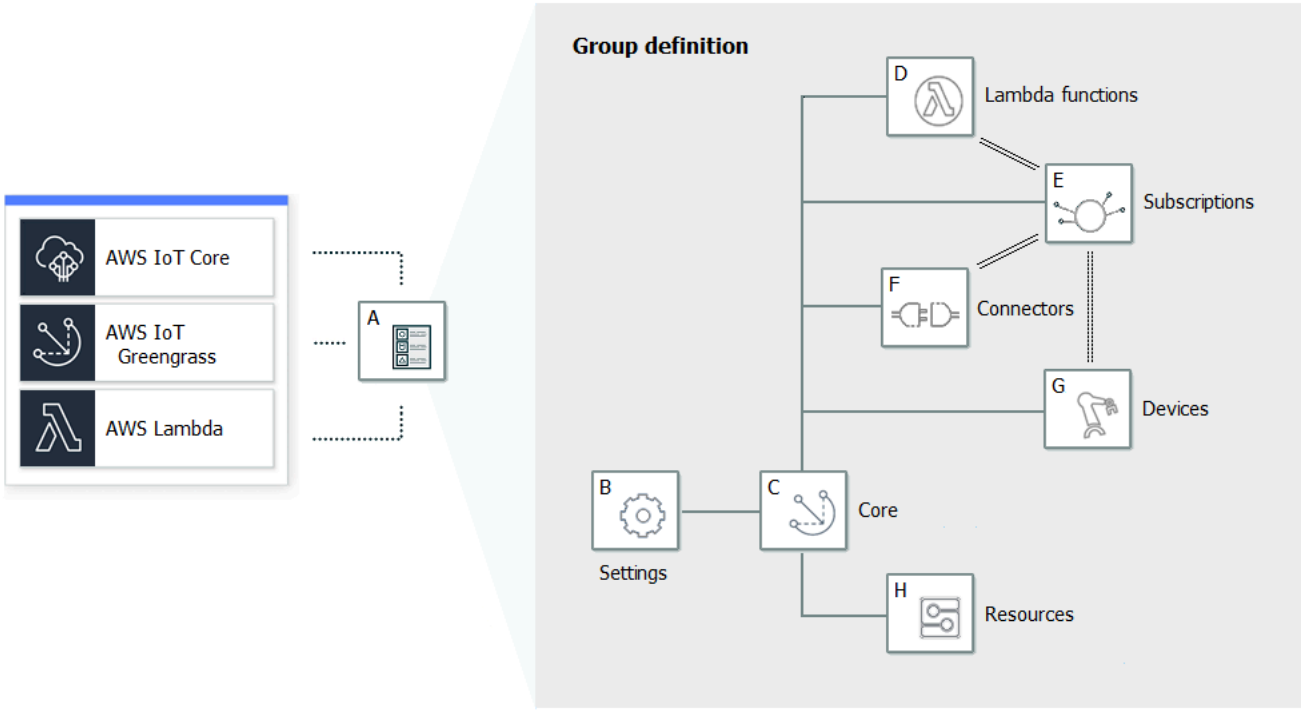
- Ejecute `/greengrass/ggc/core/greengrassd` en lugar de `/greengrass/greengrassd`.
- Implemente el grupo en el nuevo núcleo.

1.0.0

Versión inicial

AWS IoT Greengrass grupos

Un grupo de Greengrass es una colección de configuraciones y componentes, como un núcleo de Greengrass, dispositivos y suscripciones. Los grupos se utilizan para definir un ámbito de interacción. Por ejemplo, un grupo podría representar la planta de un edificio, un camión o un emplazamiento minero entero. En el siguiente diagrama se muestran los componentes que pueden componer un grupo de Greengrass.



En el diagrama anterior:

A: Definición de grupo de Greengrass

Información sobre la configuración de grupo y los componentes.

B: Configuración de grupo de Greengrass

Entre ellos se incluyen:

- Rol de grupo de Greengrass.
- Autoridad de certificación y configuración de conexión local.
- Información de conectividad del núcleo de Greengrass.
- Entorno de tiempo de ejecución predeterminado de Lambda. Para obtener más información, consulte the section called “Configuración de la creación de contenedores predeterminada para funciones Lambda de un grupo”.
- CloudWatch y configuración de registros locales. Para obtener más información, consulte the section called “Monitorización con registros de AWS IoT Greengrass”.

C: Núcleo de Greengrass

La AWS IoT cosa (dispositivo) que representa el núcleo de Greengrass. Para obtener más información, consulte the section called “Configuración de AWS IoT Greengrass Core”.

D: Definición de la función de Lambda

Una lista de las funciones de Lambda que se ejecutan localmente en el núcleo, con datos de configuración asociados. Para obtener más información, consulte Ejecutar funciones de Lambda locales .

E: Definición de suscripción

Una lista de suscripciones que permiten la comunicación mediante mensajes MQTT. Una suscripción define:

- Un origen del mensaje y un destino del mensaje. Pueden ser dispositivos cliente, funciones Lambda AWS IoT Core, conectores y el servicio paralelo local.
- Un tema o asunto que se utiliza para filtrar mensajes.

Para obtener más información, consulte the section called “Suscripciones administradas en el flujo de trabajo de mensajería de MQTT”.

F: definición de conector

Una lista de los conectores que se ejecutan localmente en el núcleo, con datos de configuración asociados. Para obtener más información, consulte Integración con servicios y protocolos mediante conectores.

G: Definición de dispositivos

Una lista de AWS IoT elementos (conocidos como dispositivos cliente o dispositivos) que son miembros del grupo Greengrass, con los datos de configuración asociados. Para obtener más información, consulte the section called “Dispositivos en AWS IoT Greengrass”.

H: Definición de recursos

Una lista de los recursos locales, recursos de aprendizaje automático y recursos de secretos en el núcleo de Greengrass, con datos de configuración asociados. Para obtener más información, consulte Acceder a recursos locales, Cómo realizar la inferencia de machine learning y Implementación de secretos en el núcleo.

Cuando se implementa, la definición del grupo de Greengrass, las funciones de Lambda, los conectores, los recursos y la tabla de suscripción se copian en el dispositivo principal. Para obtener más información, consulte Implementación de grupos de AWS IoT Greengrass.

Dispositivos en AWS IoT Greengrass

Un grupo de Greengrass puede contener dos tipos de AWS IoT dispositivos:

Núcleo de Greengrass

Un núcleo de Greengrass es un dispositivo que ejecuta el software AWS IoT Greengrass Core, lo que le permite comunicarse directamente con el servicio AWS IoT Core y el AWS IoT Greengrass mismo. Un núcleo tiene su propio certificado de dispositivo que se utiliza para autenticarse. AWS IoT Core Tiene una sombra de dispositivo y una entrada en el AWS IoT Core registro. Los núcleos Greengrass ejecutan un entorno de ejecución Lambda local, un agente de despliegue y un rastreador de direcciones IP que envía la información de la dirección IP al AWS IoT Greengrass servicio para permitir que los dispositivos cliente descubran automáticamente su información de conexión principal y de grupo. Para obtener más información, consulte the section called “Configuración de AWS IoT Greengrass Core”.

> **Note**
>
> Un grupo de Greengrass debe contener exactamente un núcleo.

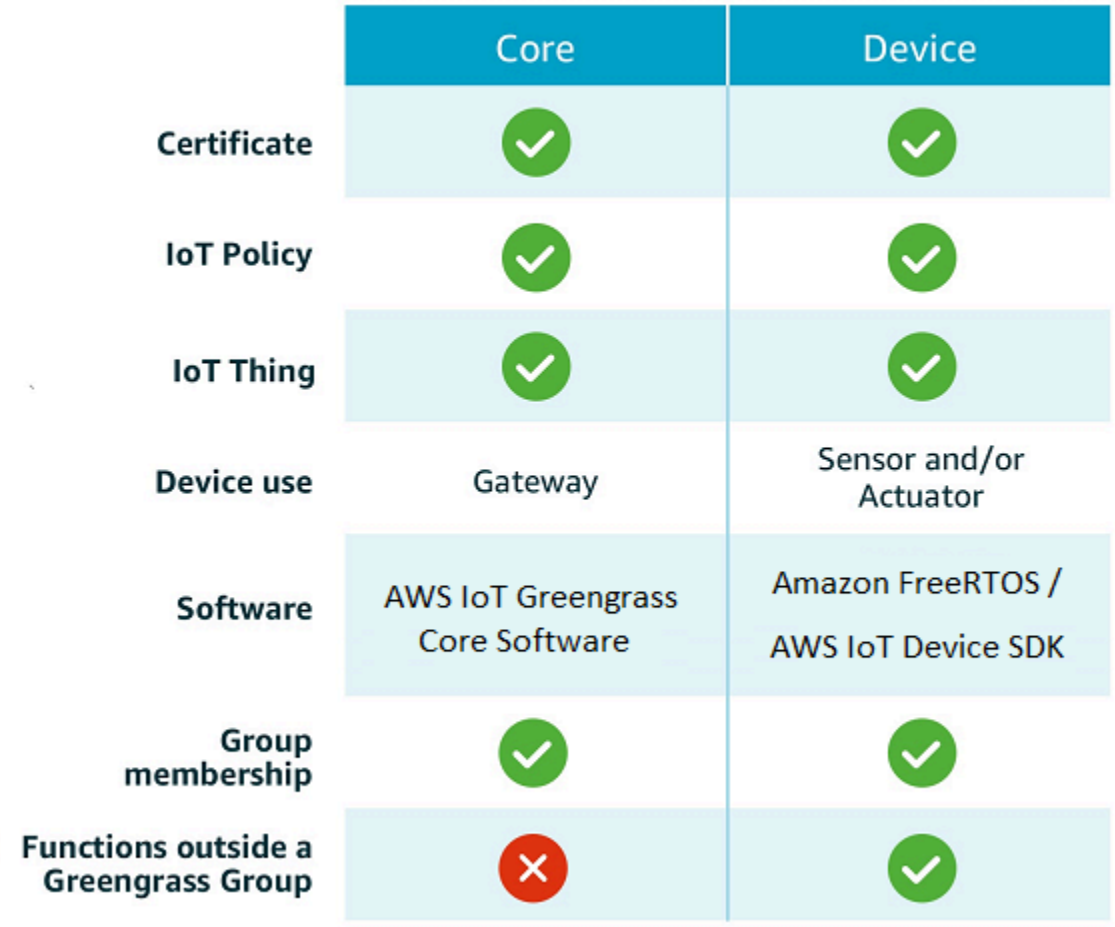
Dispositivo cliente

Los dispositivos cliente (también denominados dispositivos conectados, dispositivos Greengrass o dispositivos) son dispositivos que se conectan a un núcleo de Greengrass a través de MQTT. Tienen su propio certificado de dispositivo para la AWS IoT Core autenticación, una sombra del dispositivo y una entrada en el AWS IoT Core registro. Los dispositivos de cliente pueden ejecutar FreeRTOS o utilizar el SDK de AWS IoTo la API de AWS IoT Greengrass de descubrimiento para obtener la información de descubrimiento utilizada para conectarse y autenticarse con el núcleo en el mismo grupo de Greengrass. Para obtener información sobre cómo usar la AWS IoT consola para crear y configurar un dispositivo cliente AWS IoT Greengrass, consultethe section called “Módulo 4: Interacción con dispositivos cliente en un grupo de AWS IoT Greengrass”. O bien, si desea ver ejemplos que muestran cómo AWS CLI utilizarla para crear y configurar un dispositivo cliente AWS IoT Greengrass, consulte create-device-definitionla Referencia de AWS CLI comandos.

En un grupo de Greengrass, puede crear suscripciones que permitan a los dispositivos cliente comunicarse a través de MQTT con las funciones, conectores y otros dispositivos cliente de Lambda del grupo, y con AWS IoT Core el servicio paralelo local. Los mensajes de MQTT se enrutan a través del núcleo. Si el dispositivo de núcleo pierde la conectividad con la nube, los dispositivos de cliente pueden seguir comunicándose a través de la red local. El tamaño de estos dispositivos puede variar, desde dispositivos con microcontroladores más pequeños a dispositivos más grandes. En la actualidad, un grupo de Greengrass pueden contener un máximo de 2500 dispositivos de cliente. Un dispositivo de cliente puede ser miembro de hasta 10 grupos.

> **Note**
>
> OPC-UA es un estándar de intercambio de información para la comunicación industrial. Para implementar el soporte para OPC-UA en el núcleo de Greengrass, puede usar el conector IoT. SiteWise El conector envía datos de dispositivos industriales desde los servidores OPC-UA a las propiedades de los activos. AWS IoT SiteWise

La siguiente tabla muestra cómo se relacionan estos tipos de dispositivos.



El dispositivo AWS IoT Greengrass principal almacena los certificados en dos ubicaciones:

- Certificado del dispositivo del núcleo en `/greengrass-root/certs`. Normalmente, el certificado de dispositivo del núcleo se denomina `hash.cert.pem` (por ejemplo, `86c84488a5.cert.pem`). El AWS IoT cliente utiliza este certificado para la autenticación mutua cuando el núcleo se conecta a los AWS IoT Greengrass servicios AWS IoT Core and.
- Certificado de servidor MQTT en `/greengrass-root/ggc/var/state/server`. El certificado del servidor MQTT se denomina `server.crt`. Este certificado se utiliza para la autenticación mutua entre el servidor MQTT local (en el núcleo de Greengrass) y los dispositivos Greengrass.

> **Note**
>
> *greengrass-root* representa la ruta en la que se instala el software AWS IoT Greengrass principal en el dispositivo. Normalmente, este es el directorio `/greengrass`.

SDK

Para trabajar AWS con ellos se utilizan los siguientes SDK: AWS IoT Greengrass

AWS SDK

Utilice el AWS SDK para crear aplicaciones que interactúen con cualquier AWS servicio, incluidos Amazon S3, Amazon DynamoDB y muchos AWS IoT más. AWS IoT Greengrass En el contexto de AWS IoT Greengrass, puede usar el AWS SDK en las funciones de Lambda implementadas para realizar llamadas directas a cualquier AWS servicio. Para obtener más información, consulte SDK de AWS.

> **Note**
>
> Las operaciones específicas de Greengrass que están disponibles en los AWS SDK también están disponibles en la AWS IoT Greengrass API y. AWS CLI

AWS IoT SDK de dispositivo

El SDK para AWS IoT dispositivos ayuda a los dispositivos a conectarse AWS IoT Core y AWS IoT Greengrass. Para obtener más información, consulte los SDK de dispositivos de AWS IoT en la Guía para desarrolladores de AWS IoT .

Los dispositivos cliente pueden usar cualquiera de las plataformas AWS IoT Device SDK v2 para descubrir la información de conectividad de un núcleo de Greengrass. La información de conectividad incluye lo siguiente:

- Los ID de los grupos de Greengrass a los que pertenece el dispositivo cliente.
- Las direcciones IP del núcleo de Greengrass de cada grupo. También se denominan puntos de conexión principales.
- El certificado de CA grupal, que los dispositivos utilizan para la autenticación mutua con el núcleo. Para obtener más información, consulte the section called “Flujo de trabajo de conexión de dispositivos”.

> **Note**
>
> En la versión 1 de los SDK para AWS IoT dispositivos, solo las plataformas C++ y Python ofrecen soporte de descubrimiento integrado.

AWS IoT Greengrass SDK principal

El SDK AWS IoT Greengrass principal permite que las funciones de Lambda interactúen con el núcleo de Greengrass, publiquen mensajes e interactúen con el servicio paralelo local AWS IoT, invoquen otras funciones de Lambda desplegadas y accedan a recursos secretos. Este SDK lo utilizan las funciones de Lambda que se ejecutan en un núcleo de AWS IoT Greengrass . Para obtener más información, consulte SDK de AWS IoT Greengrass Core.

AWS IoT Greengrass SDK de Machine Learning

El AWS IoT Greengrass Machine Learning SDK permite que las funciones de Lambda consuman modelos de aprendizaje automático que se implementan en el núcleo de Greengrass como recursos de aprendizaje automático. Este SDK lo utilizan las funciones de Lambda que se ejecutan en un AWS IoT Greengrass núcleo e interactúan con un servicio de inferencia local. Para obtener más información, consulte SDK de machine learning de AWS IoT Greengrass.

Plataformas compatibles y requisitos

En las siguientes pestañas se enumeran las plataformas compatibles y los requisitos del software AWS IoT Greengrass Core.

> **Note**
>
> Puede descargar el software AWS IoT Greengrass principal desde las descargas del software AWS IoT Greengrass principal.

GGC v1.11

Plataformas admitidas:

- Arquitectura: Armv7l
  - Sistema operativo: Linux
  - Sistema operativo: Linux (OpenWrt)
- Arquitectura: Armv8 (AArch64)
  - Sistema operativo: Linux
  - Sistema operativo: Linux (OpenWrt)
- Arquitectura: Armv6l

    - Sistema operativo: Linux
- Arquitectura: x86_64
    - Sistema operativo: Linux
- Las plataformas Windows, macOS y Linux se pueden ejecutar AWS IoT Greengrass en un contenedor de Docker. Para obtener más información, consulte the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”.

Requisitos:

- Espacio en disco mínimo de 128 MB disponible para el software AWS IoT Greengrass principal. Si utiliza el agente de actualización OTA, el mínimo es de 400 MB.
- Se asigna un mínimo de 128 MB de RAM al software AWS IoT Greengrass principal. Con el administrador de secuencias habilitado, el mínimo es 198 MB de RAM.

  > **Note**
  >
  > El administrador de transmisiones está activado de forma predeterminada si utilizas la opción de creación de grupos por defecto de la AWS IoT consola para crear tu grupo de Greengrass.

- Versión del kernel de Linux:
    - Se necesita la versión 4.4 o posterior del kernel de Linux para poder ejecutar AWS IoT Greengrass contenedores.
    - Se requiere la versión 3.17 o posterior del kernel de Linux para permitir la ejecución AWS IoT Greengrass sin contenedores. En esta configuración, la creación de contenedores de la función de Lambda predeterminada para el grupo de Greengrass debe establecerse en Sin contenedor. Para ver instrucciones, consulte the section called “Configuración de la creación de contenedores predeterminada para funciones Lambda de un grupo”.
- Biblioteca C de GNU (glibc), versión 2.14 o posterior. OpenWrt Las distribuciones requieren la versión 1.1.16 o posterior de la biblioteca C de Musl.
- El directorio `/var/run` debe existir en el dispositivo.
- Los archivos `/dev/stderr`, `/dev/stdin` y `/dev/stdout` deben estar disponibles.
- La protección de enlaces permanentes y simbólicos debe estar habilitada en el dispositivo. De lo contrario, solo se AWS IoT Greengrass puede ejecutar en modo inseguro, utilizando la bandera. `-i`

- Las siguientes configuraciones de kernel de Linux deben estar habilitado en el dispositivo:
  - Espacio de nombres:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Grupos de control:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    El kernel debe admitir los grupos de control. Cuando se ejecuta AWS IoT Greengrass con contenedores, se aplican los siguientes requisitos:
    - El grupo de control (cgroup) de memoria debe estar habilitado y montado para permitir que AWS IoT Greengrass establezca el límite de memoria para las funciones de Lambda.
    - El cgroup del dispositivo debe estar habilitado y montado si se utilizan funciones Lambda con acceso a recursos locales para abrir archivos en AWS IoT Greengrass el dispositivo principal.
  - Otros:
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- El certificado raíz de Amazon S3 y AWS IoT debe estar presente en el almacén de confianza del sistema.
- El administrador de transmisiones requiere el tiempo de ejecución de Java 8 y un mínimo de 70 MB de RAM, además del requisito de memoria básico del software AWS IoT Greengrass Core. El administrador de transmisiones está activado de forma predeterminada cuando se utiliza la opción de creación de grupos predeterminada de la AWS IoT consola. El administrador de transmisiones no es compatible con las OpenWrt distribuciones.

- Las bibliotecas que admiten el tiempo de ejecución de AWS Lambda requerido por las funciones de Lambda que desea ejecutar localmente. Las bibliotecas necesarias deben instalarse en el núcleo y agregarse a la variable de entorno `PATH`. Pueden instalarse varias bibliotecas en el mismo núcleo.
  - Consulte la versión 3.8 de Python para ver las funciones que utilizan el tiempo de ejecución Python 3.8.
  - Consulte Python versión 3.7 para ver las funciones que utilizan el tiempo de ejecución Python 3.7.
  - Consulte Python versión 2.7 para ver las funciones que utilizan el tiempo de ejecución Python 2.7.
  - Node.js versión 12.x para las funciones que utilizan el entorno de tiempo de ejecución de Node.js 12.x.
  - Consulte Java versión 8 o posterior para ver las funciones que utilizan el runtime Java 8.

    > **Note**
    >
    > Oficialmente, no se admite la ejecución de Java en una OpenWrt distribución. Sin embargo, si su OpenWrt compilación es compatible con Java, es posible que pueda ejecutar funciones Lambda creadas en Java en sus dispositivos. OpenWrt

    Para obtener más información sobre la AWS IoT Greengrass compatibilidad con los tiempos de ejecución de Lambda, consulte. Ejecutar funciones de Lambda locales
- El agente de actualización (OTA) requiere los siguientes comandos de shell over-the-air (no las BusyBox variantes):
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`

- `umount`
- `mv`
- `gzip`
- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`
- `/bin/bash`

GGC v1.10

Plataformas admitidas:

- Arquitectura: Armv7l
  - Sistema operativo: Linux
  - Sistema operativo: Linux (OpenWrt)
- Arquitectura: Armv8 (AArch64)
  - Sistema operativo: Linux
  - Sistema operativo: Linux (OpenWrt)
- Arquitectura: Armv6l
  - Sistema operativo: Linux
- Arquitectura: x86_64
  - Sistema operativo: Linux
- Las plataformas Windows, macOS y Linux se pueden ejecutar AWS IoT Greengrass en un contenedor de Docker. Para obtener más información, consulte the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”.

Requisitos:

- Espacio en disco mínimo de 128 MB disponible para el software AWS IoT Greengrass principal. Si utiliza el agente de actualización OTA, el mínimo es de 400 MB.

- Se asigna un mínimo de 128 MB de RAM al software AWS IoT Greengrass principal. Con el administrador de secuencias habilitado, el mínimo es 198 MB de RAM.

  > **Note**
  >
  > El administrador de transmisiones está activado de forma predeterminada si utilizas la opción de creación de grupos por defecto de la AWS IoT consola para crear tu grupo de Greengrass.

- Versión del kernel de Linux:
  - Se necesita la versión 4.4 o posterior del kernel de Linux para poder ejecutar AWS IoT Greengrass contenedores.
  - Se requiere la versión 3.17 o posterior del kernel de Linux para permitir la ejecución AWS IoT Greengrass sin contenedores. En esta configuración, la creación de contenedores de la función de Lambda predeterminada para el grupo de Greengrass debe establecerse en Sin contenedor. Para ver instrucciones, consulte the section called “Configuración de la creación de contenedores predeterminada para funciones Lambda de un grupo”.
- Biblioteca C de GNU (glibc), versión 2.14 o posterior. OpenWrt Las distribuciones requieren la versión 1.1.16 o posterior de la biblioteca C de Musl.
- El directorio `/var/run` debe existir en el dispositivo.
- Los archivos `/dev/stderr`, `/dev/stdin` y `/dev/stdout` deben estar disponibles.
- La protección de enlaces permanentes y simbólicos debe estar habilitada en el dispositivo. De lo contrario, solo se AWS IoT Greengrass puede ejecutar en modo inseguro, utilizando la bandera. `-i`
- Las siguientes configuraciones de kernel de Linux deben estar habilitado en el dispositivo:
  - Espacio de nombres:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Grupos de control:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

El kernel debe admitir los grupos de control. Cuando se ejecuta AWS IoT Greengrass con contenedores, se aplican los siguientes requisitos:

- El grupo de control (cgroup) de memoria debe estar habilitado y montado para permitir que AWS IoT Greengrass establezca el límite de memoria para las funciones de Lambda.
- El cgroup del dispositivo debe estar habilitado y montado si se utilizan funciones Lambda con acceso a recursos locales para abrir archivos en AWS IoT Greengrass el dispositivo principal.

- Otros:
  - CONFIG_POSIX_MQUEUE
  - CONFIG_OVERLAY_FS
  - CONFIG_HAVE_ARCH_SECCOMP_FILTER
  - CONFIG_SECCOMP_FILTER
  - CONFIG_KEYS
  - CONFIG_SECCOMP
  - CONFIG_SHMEM

- El certificado raíz de Amazon S3 y AWS IoT debe estar presente en el almacén de confianza del sistema.
- El administrador de transmisiones requiere el tiempo de ejecución de Java 8 y un mínimo de 70 MB de RAM, además del requisito de memoria básico del software AWS IoT Greengrass Core. El administrador de transmisiones está activado de forma predeterminada cuando se utiliza la opción de creación de grupos predeterminada de la AWS IoT consola. El administrador de transmisiones no es compatible con las OpenWrt distribuciones.
- Las bibliotecas que admiten el tiempo de ejecución de AWS Lambda requerido por las funciones de Lambda que desea ejecutar localmente. Las bibliotecas necesarias deben instalarse en el núcleo y agregarse a la variable de entorno `PATH`. Pueden instalarse varias bibliotecas en el mismo núcleo.
  - Consulte Python versión 3.7 para ver las funciones que utilizan el tiempo de ejecución Python 3.7.
  - Consulte Python versión 2.7 para ver las funciones que utilizan el tiempo de ejecución Python 2.7.
  - Node.js versión 12.x para las funciones que utilizan el entorno de tiempo de ejecución de Node.js 12.x.

- Consulte Java versión 8 o posterior para ver las funciones que utilizan el runtime Java 8.

  > **Note**
  >
  > Oficialmente, no se admite la ejecución de Java en una OpenWrt distribución. Sin embargo, si su OpenWrt compilación es compatible con Java, es posible que pueda ejecutar funciones Lambda creadas en Java en sus dispositivos. OpenWrt

  Para obtener más información sobre la AWS IoT Greengrass compatibilidad con los tiempos de ejecución de Lambda, consulte. Ejecutar funciones de Lambda locales

- El agente de actualización (OTA) requiere los siguientes comandos de shell over-the-air (no las BusyBox variantes):
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`
  - `mkdir`
  - `rm`
  - `ln`
  - `cut`
  - `cat`
  - `/bin/bash`

GGC v1.9

Plataformas admitidas:

- Arquitectura: Armv7l
  - Sistema operativo: Linux
  - Sistema operativo: Linux (OpenWrt)
- Arquitectura: Armv8 (AArch64)
  - Sistema operativo: Linux
  - Sistema operativo: Linux (OpenWrt)
- Arquitectura: Armv6l
  - Sistema operativo: Linux
- Arquitectura: x86_64
  - Sistema operativo: Linux
- Las plataformas Windows, macOS y Linux se pueden ejecutar AWS IoT Greengrass en un contenedor de Docker. Para obtener más información, consulte the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”.

Requisitos:

- Espacio en disco mínimo de 128 MB disponible para el software AWS IoT Greengrass principal. Si utiliza el agente de actualización OTA, el mínimo es de 400 MB.
- Se asigna un mínimo de 128 MB de RAM al software AWS IoT Greengrass principal.
- Versión del kernel de Linux:
  - Se requiere la versión 4.4 o posterior del kernel de Linux para permitir la ejecución AWS IoT Greengrass con contenedores.
  - Se requiere la versión 3.17 o posterior del kernel de Linux para permitir la ejecución AWS IoT Greengrass sin contenedores. En esta configuración, la creación de contenedores de la función de Lambda predeterminada para el grupo de Greengrass debe establecerse en Sin contenedor. Para ver instrucciones, consulte the section called “Configuración de la creación de contenedores predeterminada para funciones Lambda de un grupo”.
- Biblioteca C de GNU (glibc), versión 2.14 o posterior. OpenWrt Las distribuciones requieren la versión 1.1.16 o posterior de la biblioteca C de Musl.
- El directorio `/var/run` debe existir en el dispositivo.

- Los archivos `/dev/stderr`, `/dev/stdin` y `/dev/stdout` deben estar disponibles.
- La protección de enlaces permanentes y simbólicos debe estar habilitada en el dispositivo. De lo contrario, solo se AWS IoT Greengrass puede ejecutar en modo inseguro, utilizando la bandera. `-i`
- Las siguientes configuraciones de kernel de Linux deben estar habilitado en el dispositivo:
  - Espacio de nombres:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Grupos de control:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    El kernel debe admitir los grupos de control. Cuando se ejecuta AWS IoT Greengrass con contenedores, se aplican los siguientes requisitos:
    - El grupo de control (cgroup) de memoria debe estar habilitado y montado para permitir que AWS IoT Greengrass establezca el límite de memoria para las funciones de Lambda.
    - El cgroup del dispositivo debe estar habilitado y montado si se utilizan funciones Lambda con acceso a recursos locales para abrir archivos en AWS IoT Greengrass el dispositivo principal.
  - Otros:
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- El certificado raíz de Amazon S3 y AWS IoT debe estar presente en el almacén de confianza del sistema.

- Las bibliotecas que admiten el tiempo de ejecución de AWS Lambda requerido por las funciones de Lambda que desea ejecutar localmente. Las bibliotecas necesarias deben instalarse en el núcleo y agregarse a la variable de entorno `PATH`. Pueden instalarse varias bibliotecas en el mismo núcleo.
  - Consulte Python versión 2.7 para ver las funciones que utilizan el tiempo de ejecución Python 2.7.
  - Consulte Python versión 3.7 para ver las funciones que utilizan el tiempo de ejecución Python 3.7.
  - Consulte Node.js versión 6.10 o posterior para ver las funciones que utilizan el tiempo de ejecución Node.js 6.10.
  - Consulte Node.js versión 8.10 o posterior para ver las funciones que utilizan el tiempo de ejecución Node.js 8.10.
  - Consulte Java versión 8 o posterior para ver las funciones que utilizan el runtime Java 8.

    > **Note**
    >
    > Oficialmente, no se admite la ejecución de Java en una OpenWrt distribución. Sin embargo, si su OpenWrt compilación es compatible con Java, es posible que pueda ejecutar funciones Lambda creadas en Java en sus dispositivos. OpenWrt

    Para obtener más información sobre la AWS IoT Greengrass compatibilidad con los tiempos de ejecución de Lambda, consulte. Ejecutar funciones de Lambda locales
- El agente de actualización (OTA) requiere los siguientes comandos de shell over-the-air (no las BusyBox variantes):
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`

- `umount`
- `mv`
- `gzip`
- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`

GGC v1.8

- Plataformas admitidas:
  - Arquitectura: ARMv7l; SO: Linux.
  - Arquitectura: x86_64; SO: Linux.
  - Arquitectura: Armv8 (AArch64); OS: Linux
  - Las plataformas Windows, macOS y Linux se pueden ejecutar AWS IoT Greengrass en un contenedor de Docker. Para obtener más información, consulte the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”.
  - Las plataformas Linux pueden ejecutar una versión de AWS IoT Greengrass con funciones limitadas mediante el complemento Greengrass, que está disponible a través de Snapcraft. Para obtener más información, consulte the section called “AWS IoT Greengrass software snap”.
- Se requieren los elementos siguientes:
  - Espacio en disco mínimo de 128 MB disponible para el software AWS IoT Greengrass Core. Si utiliza el agente de actualización OTA, el mínimo es de 400 MB.
  - Se asigna un mínimo de 128 MB de RAM al software AWS IoT Greengrass principal.
  - Versión del kernel de Linux:
    - Se requiere la versión 4.4 o posterior del kernel de Linux para permitir la ejecución AWS IoT Greengrass con contenedores.
    - Se requiere la versión 3.17 o posterior del kernel de Linux para permitir la ejecución AWS IoT Greengrass sin contenedores. En esta configuración, la creación de contenedores de la función de Lambda predeterminada para el grupo de Greengrass debe establecerse en

Sin contenedor. Para ver instrucciones, consulte the section called “Configuración de la creación de contenedores predeterminada para funciones Lambda de un grupo”.

- Biblioteca de C de GNU (glibc) versión 2.14 o posterior.
- El directorio `/var/run` debe existir en el dispositivo.
- Los archivos `/dev/stderr`, `/dev/stdin` y `/dev/stdout` deben estar disponibles.
- La protección de enlaces permanentes y simbólicos debe estar habilitada en el dispositivo. De lo contrario, solo se AWS IoT Greengrass puede ejecutar en modo inseguro, utilizando el `-i` indicador.
- Las siguientes configuraciones de kernel de Linux deben estar habilitado en el dispositivo:
  - Espacio de nombres:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Grupos de control:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    El kernel debe admitir los grupos de control. Cuando se ejecuta AWS IoT Greengrass con contenedores, se aplican los siguientes requisitos:

    - El grupo de control (cgroup) de memoria debe estar habilitado y montado para permitir que AWS IoT Greengrass establezca el límite de memoria para las funciones de Lambda.
    - El cgroup del dispositivo debe estar habilitado y montado si se utilizan funciones Lambda con acceso a recursos locales para abrir archivos en AWS IoT Greengrass el dispositivo principal.
  - Otros:
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER

        - CONFIG_KEYS
        - CONFIG_SECCOMP
        - CONFIG_SHMEM
    - El certificado raíz de Amazon S3 y AWS IoT debe estar presente en el almacén de confianza del sistema.
- Los elementos siguientes son obligatorios:
    - Las bibliotecas que admiten el tiempo de ejecución de AWS Lambda requerido por las funciones de Lambda que desea ejecutar localmente. Las bibliotecas necesarias deben instalarse en el núcleo y agregarse a la variable de entorno `PATH`. Pueden instalarse varias bibliotecas en el mismo núcleo.
        - Consulte Python versión 2.7 para ver las funciones que utilizan el tiempo de ejecución Python 2.7.
        - Consulte Node.js versión 6.10 o posterior para ver las funciones que utilizan el tiempo de ejecución Node.js 6.10.
        - Consulte Java versión 8 o posterior para ver las funciones que utilizan el runtime Java 8.
    - El agente de actualización (OTA) requiere los siguientes comandos de shell over-the-air (no las BusyBox variantes):
        - `wget`
        - `realpath`
        - `tar`
        - `readlink`
        - `basename`
        - `dirname`
        - `pidof`
        - `df`
        - `grep`
        - `umount`
        - `mv`
        - `gzip`
        - `mkdir`
        - `rm`

- `ln`
- `cut`
- `cat`

Para obtener información sobre AWS IoT Greengrass las cuotas (límites), consulte Service Quotas en Referencia general de Amazon Web Services.

Para obtener información sobre precios, consulte Precios de AWS IoT Greengrass y Precios de AWS IoT Core.

AWS IoT Greengrass descargas

Puede utilizar la información siguiente para buscar y descargar software para utilizarlo con AWS IoT Greengrass.

Temas

- AWS IoT Greengrass Software básico
- AWS IoT Greengrass software snap
- AWS IoT Greengrass Software Docker
- AWS IoT Greengrass SDK básico
- Bibliotecas y entornos de ejecución de aprendizaje automático compatibles
- AWS IoT Greengrass Software ML SDK

AWS IoT Greengrass Software básico

El software AWS IoT Greengrass Core extiende la AWS funcionalidad a un dispositivo AWS IoT Greengrass central, lo que permite que los dispositivos locales actúen localmente a partir de los datos que generan.

v1.11

1.11.6

Mejoras y correcciones de errores:

- Resiliencia mejorada en caso de que se produzca una pérdida repentina de energía durante una implementación.

- Se solucionó un problema por el que la corrupción de los datos del administrador de transmisiones podía impedir que se AWS IoT Greengrass iniciara el software Core.
- Se ha corregido un problema que provocaba que los nuevos dispositivos cliente no se pudieran conectar al núcleo en determinadas situaciones.
- Se ha corregido un error que provocaba que los nombres de los flujos del administrador de flujos no pudieran contener `.log`.

1.11.5

Mejoras y correcciones de errores:

- Mejoras de rendimiento generales y correcciones de errores.

1.11.4

Mejoras y correcciones de errores:

- Se ha corregido un problema con el administrador de transmisiones que impedía actualizar el software AWS IoT Greengrass Core a la versión 1.11.3. Si utilizas Stream Manager para exportar datos a la nube, ahora puedes usar una actualización OTA para actualizar una versión anterior, v1.x, del software Core a la AWS IoT Greengrass versión 1.11.4.
- Mejoras de rendimiento generales y correcciones de errores.

1.11.3

Mejoras y correcciones de errores:

- Se ha corregido un problema que provocaba que el software AWS IoT Greengrass Core que se ejecutaba en un abrir y cerrar de ojos en un dispositivo Ubuntu dejara de responder tras un repentino corte de energía en el dispositivo.
- Se ha corregido un problema que provocaba un retraso en la entrega de los mensajes MQTT a funciones de Lambda de larga duración.
- Se ha corregido un problema que provocaba que los mensajes MQTT no se enviaran correctamente cuando el valor `maxWorkItemCount` estaba establecido en un valor superior a `1024`.
- Se ha corregido un problema que provocaba que el agente de actualización de OTA ignorara el período `KeepAlive` de MQTT especificado en la propiedad `keepAlive` en `config.json`.
- Mejoras de rendimiento generales y correcciones de errores.

> **Important**
>
> Si utilizas el administrador de transmisiones para exportar datos a la nube, no actualices el software AWS IoT Greengrass Core v1.11.3 desde una versión anterior, v1.x. Si es la primera vez que habilitas Stream Manager, te recomendamos encarecidamente que instales primero la última versión del software Core. AWS IoT Greengrass

1.11.1

Mejoras y correcciones de errores:

- Se ha corregido un problema que provocaba que el administrador de flujos aumentara el uso de memoria.
- Se ha corregido un problema que provocaba que el administrador de transmisiones restableciera el número de secuencia de la transmisión `0` si el dispositivo principal de Greengrass estaba apagado durante más tiempo del período especificado time-to-live (TTL) de los datos de la transmisión.
- Se ha corregido un problema que impedía que el administrador de flujos detuviera correctamente los reintentos de exportación de datos al Nube de AWS.

1.11.0

Nuevas características:

- Un agente de telemetría del núcleo de Greengrass recopila datos de telemetría locales y los publica en. Nube de AWS Para recuperar los datos de telemetría para su posterior procesamiento, los clientes pueden crear una EventBridge regla de Amazon y suscribirse a un objetivo. Para obtener más información, consulte Recopilación de datos de telemetría del estado del sistema desde los dispositivos principales. AWS IoT Greengrass
- Una API HTTP local devuelve una instantánea del estado actual de los procesos de trabajo locales iniciados por. AWS IoT Greengrass Para obtener más información, consulte Llamadas a la API de comprobación de estado local.
- Un administrador de transmisiones exporta automáticamente los datos a Amazon S3 y AWS IoT SiteWise.

  Los nuevos parámetros del administrador de flujos le permiten actualizar las transmisiones existentes y pausar o reanudar la exportación de datos.

- Compatibilidad para ejecutar funciones Lambda de Python 3.8.x en el dispositivo principal.
- Una nueva propiedad `ggDaemonPort` en `config.json` que se usa para configurar el número de puerto IPC de Greengrass core. El número de puerto predeterminado es 8000.

  Una propiedad `systemComponentAuthTimeout` nueva en `config.json` que se utiliza para configurar el tiempo de espera de la autenticación IPC de Greengrass core. El valor de tiempo predeterminado es 5000 milisegundos.
- Se aumentó la cantidad máxima de AWS IoT dispositivos por AWS IoT Greengrass grupo de 200 a 2500.

  Se aumentó el número máximo de suscripciones por grupo de 1000 a 10 000.

  Para obtener más información, consulte Puntos de conexión y cuotas de AWS IoT Greengrass.

Mejoras y correcciones de errores:

- Optimización general que puede reducir la utilización de memoria de los procesos de servicio de Greengrass.
- Un nuevo parámetro de configuración del tiempo de ejecución (`mountAllBlockDevices`) permite a Greengrass utilizar los montajes de enlace para montar todos los dispositivos de bloques en un contenedor después de configurar el OverlayFS. Esta característica resolvió un problema que provocaba un error en la implementación de Greengrass si `/usr` no estaba por debajo de la jerarquía `/`.
- Se ha corregido un problema que provocaba un fallo en el AWS IoT Greengrass núcleo si `/tmp` se trataba de un enlace simbólico.
- Se ha corregido un error para permitir que el agente de implementación de Greengrass elimine de la carpeta los artefactos del modelo de machine learning no utilizados. `mlmodel_public`
- Mejoras de rendimiento generales y correcciones de errores.

Para instalar el software AWS IoT Greengrass Core en su dispositivo principal, descargue el paquete correspondiente a su arquitectura y sistema operativo (SO) y, a continuación, siga los pasos de la Guía de introducción.

> **Tip**
>
> AWS IoT Greengrass también ofrece otras opciones para instalar el software AWS IoT Greengrass principal. Por ejemplo, puede usar la configuración de dispositivos Greengrass para configurar su entorno e instalar la última versión del software AWS IoT Greengrass Core. O bien, en las plataformas Debian compatibles, puede utilizar el administrador de paquetes APT para instalar o actualizar el software AWS IoT Greengrass Core. Para obtener más información, consulte the section called “Instalación del software AWS IoT Greengrass Core”.

Arquitectura	Sistema operativo	Enlace
Armv8 (AArch64)	Linux	Download
Armv8 (AArch64)	Linux () OpenWrt	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Descargar
x86_64	Linux	Download

Extended life versions

1.10.5

Nuevas características en v1.10:

- Un administrador de flujos que procesa flujos de datos localmente y los exporta automáticamente a la nube de Nube de AWS . Esta característica requiere Java 8 en el dispositivo central de Greengrass. Para obtener más información, consulte Administrar secuencias de datos.
- Un nuevo conector de implementación de aplicación de Docker de Greengrass que ejecuta una aplicación Docker en un dispositivo principal. Para obtener más información, consulte the section called “Implementación de aplicaciones Docker”.

- Un nuevo SiteWise conector de IoT que envía datos de dispositivos industriales desde los servidores OPC-UA a las propiedades de los activos. AWS IoT SiteWise Para obtener más información, consulte the section called “IoT SiteWise”.
- Las funciones de Lambda que se ejecutan sin creación de contenedores pueden acceder a los recursos de machine learning del grupo Greengrass. Para obtener más información, consulte the section called “Acceso a recursos de aprendizaje automático”.
- Support para sesiones persistentes de MQTT con AWS IoT. Para obtener más información, consulte the section called “Sesiones persistentes de MQTT con AWS IoT Core”.
- El tráfico MQTT local puede desplazarse a través de un puerto distinto del puerto predeterminado 8883. Para obtener más información, consulte the section called “Puerto MQTT para la mensajería local”.
- Nuevas opciones de `queueFullPolicy` en el SDK de AWS IoT Greengrass Core para la publicación de mensajes de confianza desde funciones de Lambda.
- Compatibilidad para ejecutar funciones Lambda de Node.js 12.x en el dispositivo principal.

Mejoras y correcciones de errores:

- Las actualizaciones O ver-the-air (OTA) con integración de seguridad de hardware se pueden configurar con OpenSSL 1.1.
- El administrador de secuencias es más resistente a la corrupción de datos de archivos.
- Se ha corregido un problema que provocaba un error de montaje de sysfs en dispositivos que utilizaban el kernel de Linux 5.1 y versiones posteriores.
- Una nueva `mqttOperationTimeout` propiedad de config.json que se utiliza para establecer el tiempo de espera de las operaciones de publicación, suscripción y cancelación de la suscripción en las conexiones MQTT. AWS IoT Core
- Se ha corregido un problema que provocaba que el administrador de flujos aumentara el uso de memoria.
- Una propiedad `systemComponentAuthTimeout` nueva en `config.json` que se utiliza para configurar el tiempo de espera de la autenticación IPC de Greengrass core. El valor de tiempo predeterminado es 5000 milisegundos.
- Se ha corregido un problema que provocaba que el agente de actualización de OTA ignorara el período `KeepAlive` de MQTT especificado en la propiedad `keepAlive` en `config.json`.

- Se ha corregido un problema que provocaba que los mensajes MQTT no se enviaran correctamente cuando el valor `maxWorkItemCount` estaba establecido en un valor superior a `1024`.
- Se ha corregido un problema que provocaba un retraso en la entrega de los mensajes MQTT a funciones de Lambda de larga duración.
- Se ha corregido un problema que provocaba que el software AWS IoT Greengrass Core que se ejecutaba en un abrir y cerrar de ojos en un dispositivo Ubuntu dejara de responder tras un corte repentino de energía en el dispositivo.
- Mejoras de rendimiento generales y correcciones de errores.

Para instalar el software AWS IoT Greengrass Core en su dispositivo principal, descargue el paquete correspondiente a su arquitectura y sistema operativo (SO) y, a continuación, siga los pasos de la Guía de introducción.

Arquitectura	Sistema operativo	Enlace
Armv8 (AArch64)	Linux	Download
Armv8 (AArch64)	Linux (OpenWrt)	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Descargar
x86_64	Linux	Download

1.9.4

Nuevas características en v1.9:

- Compatibilidad con los tiempos de ejecución Lambda de Node.js 8.10 y de Python 3.7. Las funciones Lambda que utilizan los tiempos de ejecución de Python 3.7 y Node.js 8.10 ahora pueden ejecutarse en un núcleo. AWS IoT Greengrass (AWS IoT Greengrass sigue siendo compatible con los tiempos de ejecución de Python 2.7 y Node.js 6.10).

- Conexiones MQTT optimizadas. El núcleo de Greengrass establece menos conexiones con AWS IoT Core. Este cambio puede reducir los costos operativos de los cargos que se basan en el número de conexiones.
- Clave Curva elíptica (EC) para el servidor MQTT local. El servidor MQTT local admite claves EC además de las claves RSA. (El certificado del servidor MQTT tiene una firma SHA-256 RSA, independientemente del tipo de clave). Para obtener más información, consulte the section called “Entidades de seguridad”.
- Support for OpenWrt. AWS IoT Greengrass El software principal v1.9.2 o posterior se puede instalar en OpenWrt distribuciones con arquitecturas Armv8 (AArch64) y ARMv7L. OpenWrt Actualmente, no admite la inferencia de aprendizaje automático.
- Support para ARMv6L. AWS IoT Greengrass El software principal v1.9.3 o posterior se puede instalar en distribuciones de Raspbian en arquitecturas ARMv6L (por ejemplo, en dispositivos Raspberry Pi Zero).
- Actualizaciones OTA en el puerto 443 con ALPN. Los núcleos Greengrass que utilizan el puerto 443 para el tráfico MQTT ahora admiten actualizaciones de software over-the-air (OTA). AWS IoT Greengrass utiliza la extensión TLS de Application Layer Protocol Network (ALPN) para habilitar estas conexiones. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core y the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.

Para instalar el software AWS IoT Greengrass principal en su dispositivo principal, descargue el paquete correspondiente a su arquitectura y sistema operativo (SO) y, a continuación, siga los pasos de la Guía de introducción.

Arquitectura	Sistema operativo	Enlace
Armv8 (AArch64)	Linux	Download
Armv8 (AArch64)	Linux (OpenWrt)	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Descargar
x86_64	Linux	Download

1.8.4

- Nuevas características:
  - Identidad de acceso predeterminada configurable para las funciones de Lambda del grupo. Esta configuración de nivel de grupo determina los permisos predeterminados que se utilizan para ejecutar funciones de Lambda. Puede definir el ID de usuario, el ID de grupo o ambos. Las funciones de Lambda pueden invalidar la identidad de acceso predeterminada de su grupo. Para obtener más información, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.
  - Tráfico HTTPS a través del puerto 443. La comunicación HTTPS se puede configurar para la comunicación a través del puerto 443 en lugar del puerto 8443 predeterminado. Esto complementa la AWS IoT Greengrass compatibilidad con la extensión TLS de Application Layer Protocol Network (ALPN) y permite que todo el tráfico de mensajería de Greengrass, tanto MQTT como HTTPS, utilice el puerto 443. Para obtener más información, consulte the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.
  - Identificadores AWS IoT de cliente con nombres predecibles para las conexiones. Este cambio proporciona compatibilidad con los AWS IoT Device Defender y eventos de ciclo de vida de AWS IoT, para que puede recibir notificaciones de eventos de conexión, desconexión, suscripción y anulación de suscripción. Los nombres predecibles también permiten crear lógica en torno a los ID de conexión (por ejemplo, crear plantillas de política de suscripción basadas en atributos del certificado). Para obtener más información, consulte the section called “ID de cliente para conexiones MQTT con AWS IoT”.

  Mejoras y correcciones de errores:

  - Se ha corregido un problema con la sincronización de sombras y la reconexión del administrador de certificados de dispositivo.
  - Mejoras de rendimiento generales y correcciones de errores.

Para instalar el software AWS IoT Greengrass principal en su dispositivo principal, descargue el paquete correspondiente a su arquitectura y sistema operativo (SO) y, a continuación, siga los pasos de la Guía de introducción.

Arquitectura	Sistema operativo	Enlace
Armv8 (AArch64)	Linux	Download
Armv7l	Linux	Descargar
x86_64	Linux	Download

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

Para obtener información sobre otras opciones para instalar el software AWS IoT Greengrass Core en el dispositivo, consultethe section called “Instalación del software AWS IoT Greengrass Core”.

AWS IoT Greengrass software snap

AWS IoT Greengrass snap 1.11.x le permite ejecutar una versión limitada a AWS IoT Greengrass través de cómodos paquetes de software, junto con todas las dependencias necesarias, en un entorno contenerizado.

> **Note**
>
> La AWS IoT Greengrass instantánea está disponible para la versión 1.11.x del software Core. AWS IoT Greengrass AWS IoT Greengrass no proporciona una instantánea para la versión 1.10.x. Las versiones que no son compatibles no reciben actualizaciones ni correcciones de errores.
>
> La AWS IoT Greengrass instantánea no admite conectores ni la inferencia de aprendizaje automático (ML).

Para obtener más información, consulte the section called “Ejecución de AWS IoT Greengrass en un snap”.

AWS IoT Greengrass Software Docker

AWS proporciona un Dockerfile e imágenes de Docker que facilitan su ejecución AWS IoT Greengrass en un contenedor Docker.

Dockerfile

Los archivos Docker contienen código fuente para crear imágenes de contenedores personalizadas. AWS IoT Greengrass Las imágenes se pueden modificar para ejecutarlas en arquitecturas de plataforma distintas o para reducir su tamaño. Consulte el archivo README para obtener instrucciones.

Descargue la versión de software AWS IoT Greengrass Core de destino.

v1.11

- Dockerfile para AWS IoT Greengrass la versión 1.11.6.

Extended life versions

v1.10

Dockerfile para la AWS IoT Greengrass versión 1.10.5.

v1.9

Dockerfile para la versión 1.9.4. AWS IoT Greengrass

v1.8

Dockerfile para la versión 1.8.1. AWS IoT Greengrass

Imagen de Docker

Las imágenes de Docker tienen el software AWS IoT Greengrass principal y las dependencias instaladas en las imágenes base de Amazon Linux 2 (x86_64) y Alpine Linux (x86_64, ARMv7L o AArch64). Puede usar imágenes prediseñadas para comenzar a experimentar con AWS IoT Greengrass.

> **Important**
>
> El 30 de junio de 2022, AWS IoT Greengrass finalizó el mantenimiento de las imágenes de Docker de AWS IoT Greengrass Core software v1.x publicadas en Amazon Elastic

Container Registry (Amazon ECR) y Docker Hub. Puede seguir descargando estas imágenes de Docker desde Amazon ECR y Docker Hub hasta el 30 de junio de 2023, es decir, un año después de que finalice el mantenimiento. Sin embargo, las imágenes de Docker de la versión 1.x del software AWS IoT Greengrass principal ya no reciben parches de seguridad ni correcciones de errores una vez finalizado el mantenimiento el 30 de junio de 2022. Si ejecuta una carga de trabajo de producción que depende de estas imágenes de Docker, le recomendamos que cree sus propias imágenes de Docker con los archivos Docker que se proporcionan. AWS IoT Greengrass Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

Descargue una imagen prediseñada de Docker Hub o Amazon Elastic Container Registry (Amazon ECR).

- En el caso de Docker Hub, utilice la etiqueta de *`versión`* para descargar una versión específica de la imagen de Docker de Greengrass. Para buscar etiquetas para todas las imágenes disponibles, consulte la página Tags (Etiquetas) en Docker Hub.
- En el caso de Amazon ECR, utilice la etiqueta `latest` para descargar la última versión disponible de la imagen de Docker de Greengrass. Para obtener más información sobre la publicación de las versiones de imágenes disponibles y la descarga de imágenes de Amazon ECR, consulte Ejecución de AWS IoT Greengrass en un contenedor Docker.

Warning

A partir de la versión 1.11.6 del software AWS IoT Greengrass Core, las imágenes de Docker de Greengrass ya no incluyen Python 2.7, porque Python 2.7 llegó end-of-life en 2020 y ya no recibe actualizaciones de seguridad. Si decide actualizar a estas imágenes de Docker, le recomendamos que compruebe que sus aplicaciones funcionan con las nuevas imágenes de Docker antes de implementar las actualizaciones en los dispositivos de producción. Si necesita Python 2.7 para su aplicación que usa una imagen de Docker de Greengrass, puede modificar el Dockerfile de Greengrass para incluir Python 2.7 en su aplicación.

AWS IoT Greengrass no proporciona imágenes de Docker para la versión 1.11.1 del software Core. AWS IoT Greengrass

> **Note**
>
> De forma predeterminada, las imágenes `alpine-aarch64` y `alpine-armv7l` solo pueden ejecutarse en hosts basados en Arm. Para ejecutar estas imágenes en un host x86, puede instalar QEMU y montar las bibliotecas QEMU en el host. Por ejemplo:
>
> ```
> docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
> ```

AWS IoT Greengrass SDK básico

Las funciones Lambda utilizan el SDK AWS IoT Greengrass principal para interactuar con el AWS IoT Greengrass núcleo de forma local. Esto permite a las funciones de Lambda implementadas:

- Intercambia mensajes MQTT con. AWS IoT Core
- Intercambiar mensajes MQTT con conectores, dispositivos de cliente y otras funciones de Lambda en el grupo de Greengrass.
- Interactuar con el servicio de sombra local.
- Invocar otras funciones de Lambda locales.
- Tener acceso a recursos de secretos.
- Interactuar con el administrador de secuencias.

Descargue el SDK AWS IoT Greengrass principal para su idioma o plataforma desde GitHub.

- AWS IoT Greengrass Core SDK for Java
- AWS IoT Greengrass SDK básico para Node.js
- AWS IoT Greengrass SDK básico para Python
- AWS IoT Greengrass SDK básico para C

Para obtener más información, consulte SDK de AWS IoT Greengrass Core.

Bibliotecas y entornos de ejecución de aprendizaje automático compatibles

Para realizar la inferencia en un núcleo de Greengrass, debe instalar la biblioteca o el entorno de ejecución de aprendizaje automático apropiados para el tipo de modelo de aprendizaje automático.

AWS IoT Greengrass admite los siguientes tipos de modelos de ML. Utilice estos vínculos para buscar información acerca de cómo instalar la biblioteca o el entorno de ejecución apropiados para el tipo de modelo y la plataforma del dispositivo.

- Tiempo de ejecución de aprendizaje profundo (DLR)
- MXNet
- TensorFlow

Ejemplos de aprendizaje automático

AWS IoT Greengrass proporciona ejemplos que puede utilizar con bibliotecas y tiempos de ejecución de ML compatibles. Estos ejemplos se publican bajo el contrato de licencia de software de Greengrass Core.

Deep learning runtime (DLR)

Descargue el ejemplo apropiado para la plataforma de su dispositivo:

- Ejemplo de DLR para Raspberry Pi
- Ejemplo de DLR para NVIDIA Jetson TX2
- Ejemplo de DLR para Intel Atom

Para ver un tutorial en el que se utiliza el ejemplo de DLR, consulte the section called “Cómo configurar la inferencia de machine learning optimizado”.

MXNet

Descargue el ejemplo apropiado para la plataforma de su dispositivo:

- Ejemplo de MXNet para Raspberry Pi
- Ejemplo de MXNet para NVIDIA Jetson TX2
- Ejemplo de MXNet para Intel Atom

Para ver un tutorial en el que se utiliza el ejemplo de MXNet, consulte the section called “Cómo configurar la inferencia de machine learning”.

TensorFlow

Descargue el ejemplo de Tensorflow para la plataforma de su dispositivo. Este ejemplo funciona con Raspberry Pi, NVIDIA Jetson TX2 e Intel Atom.

AWS IoT Greengrass Software ML SDK

SDK de machine learning de AWS IoT Greengrass permite que las funciones de Lambda que autorice consuman un modelo de machine learning local y envíen datos al conector ML Feedback para cargarlos y publicarlos.

v1.1.0

- Python 3.7

v1.0.0

- Python 2.7.

Esperamos tener noticias suyas

Agradecemos sus comentarios. Para contactarnos, visite AWS re:Post y use la etiqueta AWS IoT Greengrass.

Instalación del software AWS IoT Greengrass Core

El software de AWS IoT Greengrass Core lleva la funcionalidad de AWS a un dispositivo de AWS IoT Greengrass, lo que permite a los dispositivos locales actuar localmente a partir de los datos que generan.

AWS IoT Greengrass proporciona varias opciones para instalar el software de AWS IoT Greengrass Core:

- Descargar y extraer un archivo tar.gz.

- Ejecutar el script de configuración de dispositivos Greengrass.
- Instalar desde un repositorio APT.

AWS IoT Greengrass también proporciona entornos en contenedores que ejecutan el software de AWS IoT Greengrass Core.

- Ejecutar AWS IoT Greengrass en un contenedor de Docker.
- Ejecutar AWS IoT Greengrass en un snap.

Descargar y extraer el paquete del software de AWS IoT Greengrass Core

Elija el software de AWS IoT Greengrass Core para su plataforma para descargar como un archivo tar.gz y extraer en su dispositivo. Puede descargar versiones recientes del software. Para obtener más información, consulte the section called “AWS IoT Greengrass Software básico”.

Ejecutar el script de configuración de dispositivos Greengrass

Ejecute la configuración de Greengrass para configurar el dispositivo, instale la versión más reciente del software AWS IoT Greengrass Core e implemente una función Hello World de Lambda en cuestión de minutos. Para obtener más información, consulte the section called “Inicio rápido: configuración del dispositivo Greengrass”.

Instalación del software AWS IoT Greengrass Core desde un repositorio de APT

> **Important**
>
> A partir del 11 de febrero de 2022, ya no podrá instalar ni actualizar el software AWS IoT Greengrass Core desde un repositorio de APT. En los dispositivos en los que haya agregado el repositorio AWS IoT Greengrass, debe eliminarlo de la lista de fuentes. Los dispositivos que ejecutan el software desde el repositorio de APT seguirán funcionando con normalidad.

Le recomendamos que actualice el software de AWS IoT Greengrass Core mediante archivos tar.

El repositorio APT proporcionado por AWS IoT Greengrass incluye los siguientes paquetes:

- `aws-iot-greengrass-core`. Instala el software de AWS IoT Greengrass Core.
- `aws-iot-greengrass-keyring`. Instala las claves GnuPG (GPG) utilizadas para firmar el repositorio de paquetes de AWS IoT Greengrass.

  Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

Temas

- Uso de scripts de systemd para administrar el ciclo de vida del demonio de Greengrass
- Desinstale el software AWS IoT Greengrass Core mediante el repositorio APT
- Elimine las fuentes del repositorio del software AWS IoT Greengrass Core

Uso de scripts de systemd para administrar el ciclo de vida del demonio de Greengrass

El paquete `aws-iot-greengrass-core` también instala scripts de `systemd` que puede utilizar para administrar el ciclo de vida del software de AWS IoT Greengrass (demonio).

- Para iniciar el demonio de Greengrass durante el arranque:

```
systemctl enable greengrass.service
```

- Para iniciar el demonio de Greengrass:

```
systemctl start greengrass.service
```

- Para detener el demonio de Greengrass:

```
systemctl stop greengrass.service
```

- Para comprobar el estado del demonio de Greengrass:

```
systemctl status greengrass.service
```

Desinstale el software AWS IoT Greengrass Core mediante el repositorio APT

Al desinstalar el software AWS IoT Greengrass Core, puede elegir si desea conservar o eliminar la información de configuración del software AWS IoT Greengrass Core, como los certificados de los dispositivos, la información de los grupos y los archivos de registro.

Para desinstalar el software AWS IoT Greengrass Core y conservar la información de configuración

- Ejecute el siguiente comando para eliminar los paquetes de software AWS IoT Greengrass Core y conservar la información de configuración en la carpeta `/greengrass`.

  ```
  sudo apt remove aws-iot-greengrass-core aws-iot-greengrass-keyring
  ```

Para desinstalar el software AWS IoT Greengrass core y conservar la información de configuración

1. Ejecute el siguiente comando para eliminar los paquetes de software AWS IoT Greengrass Core y eliminar la información de configuración de `/greengrass folder`.

   ```
   sudo apt purge aws-iot-greengrass-core aws-iot-greengrass-keyring
   ```
2. Elimine el repositorio de software AWS IoT Greengrass Core de su lista de fuentes. Para obtener más información, consulte Elimine las fuentes del repositorio del software AWS IoT Greengrass Core.

Elimine las fuentes del repositorio del software AWS IoT Greengrass Core

Puede eliminar las fuentes del repositorio del software AWS IoT Greengrass Core cuando ya no necesite instalar o actualizar el software AWS IoT Greengrass Core del repositorio de APT. Después del 11 de febrero de 2022, debe eliminar el repositorio de su lista de fuentes para evitar que se produzca un error al ejecutar `apt update`.

Para eliminar el repositorio de APT de la lista de fuentes

- Ejecute los siguientes comandos para eliminar el repositorio del software AWS IoT Greengrass Core de la lista de fuentes.

```
sudo rm /etc/apt/sources.list.d/greengrass.list
sudo apt update
```

Ejecutar AWS IoT Greengrass en un contenedor de Docker

AWS IoT Greengrass proporciona un Dockerfile e imágenes de Docker que le facilitan la ejecución del software de AWS IoT Greengrass Core en un contenedor de Docker. Para obtener más información, consulte the section called “AWS IoT Greengrass Software Docker”.

> **Note**
>
> También puede ejecutar una aplicación de Docker en un dispositivo central de Greengrass. Para ello, utilice el Conector de implementación de la aplicación Greengrass Docker.

Ejecución de AWS IoT Greengrass en un snap

AWS IoT Greengrass snap 1.11.x le permite ejecutar una versión limitada de AWS IoT Greengrass mediante cómodos paquetes de software, junto con todas las dependencias necesarias, en un entorno contenerizado.

El 31 de diciembre de 2023, AWS IoT Greengrass finalizará el mantenimiento de la versión 1.11.x del software AWS IoT Greengrass Core Snap, publicada en snapcraft.io. Los dispositivos que ejecutan actualmente el Snap seguirán funcionando hasta nuevo aviso. Sin embargo, el núcleo de AWS IoT Greengrass de Snap ya no recibirá parches de seguridad ni correcciones de errores una vez finalizado el mantenimiento.

Conceptos de Snap

Los siguientes son conceptos básicos de Snap que le ayudarán a entender cómo usar el snap AWS IoT Greengrass:

Canal

Un componente de snap que define qué versión de un snap se instala y se le realiza el seguimiento de las actualizaciones. Los snaps se actualizan automáticamente a la última versión del canal actual.

Interfaz

Un componente de snap que permite el acceso a los recursos, como las redes y los archivos de los usuarios.

Para ejecutar el snap AWS IoT Greengrass, deben estar conectadas las siguientes interfaces. Tenga en cuenta que `greengrass-support-no-container` debe conectarse primero y nunca desconectarse.

```
- greengrass-support-no-container
- hardware-observe
- home-for-hooks
- hugepages-control
- log-observe
- mount-observe
- network
- network-bind
- network-control
- process-control
- system-observe
```

Todas las demás interfaces son opcionales. Si las funciones de Lambda requieren acceso a recursos específicos, es posible que necesite conectarse a las interfaces adecuadas.

Actualizar

Los snaps se actualizan automáticamente. De forma predeterminada, el snap daemon `snapd` es el administrador de paquetes de snap que comprueba si hay actualizaciones cuatro veces al día. Cada comprobación de actualizaciones se denomina actualización. Cuando se produce una actualización, daemon se detiene, el snap se actualiza y, a continuación, daemon se reinicia.

Para obtener más información, consulte el sitio web de Snapcraft.

Novedades de snap versión 1.11.x AWS IoT Greengrass

A continuación, se describen las novedades y los cambios de la versión 1.11.x de snap AWS IoT Greengrass.

- Esta versión solo es compatible con el usuario `snap_daemon`, que se muestra como ID de usuario (UID) y de grupo (GID) `584788`.
- Esta versión solo admite funciones de Lambda no organizadas en contenedores.

> **Important**
>
> Como las funciones de Lambda no organizadas en contenedores deben compartir el mismo usuario (`snap_daemon`), las funciones de Lambda no están aisladas entre sí. Para obtener más información, consulte Control de la ejecución de las funciones de Lambda de Greengrass mediante a configuración específica del grupo.

- Esta versión es compatible con los tiempos de ejecución C, C++, Java 8, Node.js 12.x, Python 2.7, Python 3.7 y Python 3.8.

> **Note**
>
> Para evitar tiempos de ejecución redundantes de Python, las funciones de Lambda de Python 3.7 en realidad ejecutan el tiempo de ejecución de Python 3.8.

Introducción al snap de AWS IoT Greengrass

El siguiente procedimiento le ayuda a instalar y configurar el snap AWS IoT Greengrass en su dispositivo.

Requisitos

Para ejecutar el snap de AWS IoT Greengrass, debe hacer lo siguiente:

- Ejecute el snap AWS IoT Greengrass en una distribución de Linux compatible, como Ubuntu, Linux Mint, Debian y Fedora.
- Instale daemon `snapd` en su dispositivo. Daemon `snapd`, incluida la herramienta `snap`, administra el entorno de snap en su dispositivo.

Para ver la lista de distribuciones de Linux compatibles y las instrucciones de instalación, consulte Instalación de snapd en la documentación de Snap.

Instalación y configuración del snap de AWS IoT Greengrass

En el siguiente tutorial, se muestran los primeros pasos para instalar y configurar el snap AWS IoT Greengrass en su dispositivo.

> **Note**
>
> - Aunque en este tutorial se utiliza una instancia de Amazon EC2 (x86 t2.micro Ubuntu 20.04), puede ejecutar el snap AWS IoT Greengrass con hardware físico, como una Raspberry Pi.
> - Daemon `snapd` viene preinstalado en Ubuntu.

1. Instale el snap de `core18` ejecutando el siguiente comando en el terminal del dispositivo:

   ```
   sudo snap install core18
   ```

   El snap `core18` es una snap básico que proporciona un entorno de tiempo de ejecución con bibliotecas de uso frecuente. Este snap se ha creado a partir de Ubuntu 18.04 LTS.

2. Actualice `snapd` ejecutando el siguiente comando:

   ```
   sudo snap install --channel=edge snapd; sudo snap refresh --channel=edge snapd
   ```

3. Ejecute el comando `snap list` para comprobar si tiene el snap AWS IoT Greengrass instalado.

   El siguiente ejemplo de respuesta muestra que `snapd` está instalado, pero `aws-iot-greengrass` no lo está.

   ```
   Name              Version              Rev    Tracking         Publisher   Notes
   amazon-ssm-agent  3.0.161.0            2996   latest/stable/…  aws#        classic
   core              16-2.48              10444  latest/stable    canonical#  core
   core18            20200929             1932   latest/stable    canonical#  base
   lxd               4.0.4                18150  4.0/stable/…     canonical#  -
   snapd             2.48+git548.g929ccfb 10526  latest/edge      canonical#  snapd
   ```

4. Elija una de las siguientes opciones para instalar el snap AWS IoT Greengrass 1.11.x.

   - Ejecute el siguiente siguiente comando para instalar el snap de AWS IoT Greengrass.

     ```
     sudo snap install aws-iot-greengrass
     ```

     Respuesta de ejemplo:

     ```
     aws-iot-greengrass 1.11.5 from Amazon Web Services (aws) installed
     ```

   - Para migrar de una versión anterior a la versión 1.11.x o actualizar a la última versión de parche disponible, ejecute el siguiente comando:

     ```
     sudo snap refresh --channel=1.11.x aws-iot-greengrass
     ```

   Al igual que otros snaps, el snap AWS IoT Greengrass usa canales para administrar las versiones secundarias. Los snaps se actualizan automáticamente a la última versión del canal actual. Por ejemplo, si especifica `--channel=1.11.x`, su snap AWS IoT Greengrass se actualiza a la versión 1.11.5.

   Puede ejecutar el comando `snap info aws-iot-greengrass` para obtener la lista de canales disponibles para AWS IoT Greengrass.

   Respuesta de ejemplo:

   ```
   name:      aws-iot-greengrass
   summary:   AWS supported software that extends cloud capabilities to local devices.
   publisher: Amazon Web Services (aws#)
   store-url: https://snapcraft.io/aws-iot-greengrass
   contact:   https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass
   license:   Proprietary
   description: |
     AWS IoT Greengrass seamlessly extends AWS onto edge devices so they can act
    locally on the data
     they generate, while still using the cloud for management, analytics, and durable
    storage.
     AWS IoT Greenrgrass snap v1.11.0 enables you to run a limited version of AWS IoT
    Greengrass with
     all necessary dependencies in a containerized environment.
   ```

```
  The AWS IoT Greengrass snap doesn't support connectors and machine learning (ML)
 inference.
  By downloading this software you agree to the Greengrass Core Software License
 Agreement
  (https://s3-us-west-2.amazonaws.com/greengrass-release-license/greengrass-
license-v1.pdf).
  For more information, see Run AWS IoT Greengrass in a snap
  (https://docs.aws.amazon.com/greengrass/latest/developerguide/install-
ggc.html#gg-snap-support) in
  the AWS IoT Greengrass Developer.
  If you need help, try the AWS IoT Greengrass tag on AWS re:Post
  (https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass) or connect
 with an AWS IQ expert
  (https://iq.aws.amazon.com/services/aws/greengrass).
snap-id: SRDuhPJGj4XPxFNNZQKOTvURAp0wxKnd
channels:
  latest/stable:    1.11.3 2021-06-15 (59) 111MB -
  latest/candidate: 1.11.3 2021-06-14 (59) 111MB -
  latest/beta:      1.11.3 2021-06-14 (59) 111MB -
  latest/edge:      1.11.3 2021-06-14 (59) 111MB -
  1.11.x/stable:    1.11.3 2021-06-15 (59) 111MB -
  1.11.x/candidate: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/beta:      1.11.3 2021-06-15 (59) 111MB -
  1.11.x/edge:      1.11.3 2021-06-15 (59) 111MB -
```

5. Para acceder a los recursos específicos que necesitan sus funciones de Lambda, puede conectarse a interfaces adicionales.

   Ejecute el siguiente comando para obtener la lista de interfaces compatibles con el snap AWS IoT Greengrass:

   ```
   snap connections aws-iot-greengrass
   ```

   Respuesta de ejemplo:

   ```
   Interface               Plug                                    Slot
               Notes
   camera                  aws-iot-greengrass:camera               -
               -
   dvb                     aws-iot-greengrass:dvb                  -
               -
   ```

```
gpio                    aws-iot-greengrass:gpio                              -
               -
gpio-memory-control     aws-iot-greengrass:gpio-memory-control               -
               -
greengrass-support      aws-iot-greengrass:greengrass-support-no-container
  :greengrass-support  -
hardware-observe        aws-iot-greengrass:hardware-observe
  :hardware-observe    manual
hardware-random-control  aws-iot-greengrass:hardware-random-control          -
               -
home                    aws-iot-greengrass:home-for-greengrassd              -
               -
home                    aws-iot-greengrass:home-for-hooks                    :home
               manual
hugepages-control       aws-iot-greengrass:hugepages-control
  :hugepages-control   manual
i2c                     aws-iot-greengrass:i2c                               -
               -
iio                     aws-iot-greengrass:iio                               -
               -
joystick                aws-iot-greengrass:joystick                          -
               -
log-observe             aws-iot-greengrass:log-observe                       :log-
observe         manual
mount-observe           aws-iot-greengrass:mount-observe
  :mount-observe       manual
network                 aws-iot-greengrass:network
  :network             -
network-bind            aws-iot-greengrass:network-bind
  :network-bind        -
network-control         aws-iot-greengrass:network-control
  :network-control     -
opengl                  aws-iot-greengrass:opengl
  :opengl              -
optical-drive           aws-iot-greengrass:optical-drive
  :optical-drive       -
process-control         aws-iot-greengrass:process-control
  :process-control     -
raw-usb                 aws-iot-greengrass:raw-usb                           -
               -
removable-media         aws-iot-greengrass:removable-media                   -
               -
serial-port             aws-iot-greengrass:serial-port                       -
               -
```

```
spi                       aws-iot-greengrass:spi                                -
               -
system-observe            aws-iot-greengrass:system-observe
  :system-observe      -
```

Si ve un guion (-) en la columna Slot (Ranura), la interfaz correspondiente no está conectada.

6. Siga Instalación del software AWS IoT Greengrass Core para crear un objeto AWS IoT, un grupo de Greengrass, recursos de seguridad que permitan una comunicación segura con AWS IoT y el archivo de configuración del software de AWS IoT Greengrass Core. El archivo de configuración, `config.json`, contiene la configuración específica de su núcleo de Greengrass, como la ubicación de los archivos de certificado y el punto de enlace de datos del dispositivo AWS IoT.

   > **Note**
   >
   > Si descargó el archivo en un dispositivo diferente, siga este paso para transferir los archivos al dispositivo del núcleo AWS IoT Greengrass.

7. Para el snap AWS IoT Greengrass, asegúrese de actualizar el archivo config.json, como se muestra a continuación:

   - Reemplace cada instancia de *`certificateId`* por la identificación del certificado en el nombre de los archivos de certificado y clave.
   - Si ha descargado un certificado de CA raíz de Amazon diferente al de Amazon Root CA 1, sustituya cada instancia de *`AmazonRootCA1.pem`* por el nombre del archivo de CA raíz de Amazon.

```
{
  ...
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.keyy"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.key",
        "certificatePath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-certificate.pem.crt"
```

```
      }
    },
    "caPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/AmazonRootCA1.pem"
  },
  "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
  "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

8. Ejecute el siguiente comando para añadir sus certificados de AWS IoT Greengrass y archivos de configuración:

```
sudo snap set aws-iot-greengrass gg-certs=/home/ubuntu/my-certs
```

Despliegue de una función de Lambda

En esta sección, se muestra cómo implementar una función de Lambda gestionada por el cliente en el snap AWS IoT Greengrass.

> **Important**
>
> La versión 1.11 del snap de AWS IoT Greengrass solo admite funciones de Lambda no organizadas en contenedores.

1. Ejecute el siguiente comando para iniciar el daemon de AWS IoT Greengrass.

```
sudo snap start aws-iot-greengrass
```

Respuesta de ejemplo:

```
Started.
```

> **Note**
>
> Si aparece un error, puede usar el comando `snap run` para obtener un mensaje de error detallado. Para obtener más información sobre solución de problemas, consulte error: no se pueden realizar las siguientes tareas: - Ejecute el comando de servicio «start» para los servicios ["greengrassd"] de snap "" ([start snap. aws-iot-

greengrass aws-iot-greengrass.greengrassd.service] falló con el estado de salida 1: Job for snap. aws-iot-greengrass.greengrassd.service falló porque el proceso de control finalizó con un código de error. Consulte «resumen de estado de systemctl». aws-iot-greengrass.greengrassd.service y journalctl -xe para obtener más información.) .

2. Ejecute el siguiente comando para confirmar que el daemon se está ejecutando:

```
snap services aws-iot-greengrass.greengrassd
```

Respuesta de ejemplo:

```
Service                          Startup   Current  Notes
aws-iot-greengrass.greengrassd  disabled  active   -
```

3. Siga el módulo 3 (parte 1): funciones de Lambda en AWS IoT Greengrass para crear e implementar una función de Lambda de Hello World. Sin embargo, antes de implementar la función de Lambda, complete el siguiente paso.
4. Asegúrese de que la función de Lambda se ejecute como usuario `snap_daemon` y en modo sin contenedor. Para actualizar la configuración de su grupo de Greengrass, haga lo siguiente en la consola de AWS IoT Greengrass:
   a. Inicie sesión en la consola de AWS IoT Greengrass.
   b. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
   c. En Grupos de Greengrass, elija su grupo objetivo.
   d. En la página de configuración del grupo, en el panel de navegación, elija la pestaña Funciones de lambda
   e. En Entorno de tiempo de ejecución de funciones de Lambda predeterminado, elija Editar, y haga lo siguiente:
      i. En Usuario y grupo del sistema predeterminados, elija Otro ID de usuario/ID de grupo y, a continuación, introduzca **584788** tanto para ID de usuario del sistema (número) como para ID de grupo del sistema (número).
      ii. Para la creación de contenedores predeterminada de la función de Lambda, elija Sin contenedor.
      iii. Seleccione Guardar.

Cómo detener daemon AWS IoT Greengrass

Puede utilizar el comando `snap stop` para detener un servicio. .

Ejecute el siguiente comando para detener el daemon de AWS IoT Greengrass:

```
sudo snap stop aws-iot-greengrass
```

El comando debe devolver `Stopped`..

Para comprobar si el snap se detuvo correctamente, ejecute el siguiente comando:

```
snap services aws-iot-greengrass.greengrassd
```

Respuesta de ejemplo:

```
Service                         Startup   Current   Notes
aws-iot-greengrass.greengrassd  disabled  inactive  -
```

Desinstalación del snap AWS IoT Greengrass

Ejecute el siguiente siguiente comando para desinstalar el snap de AWS IoT Greengrass.

```
sudo snap remove aws-iot-greengrass
```

Respuesta de ejemplo:

```
aws-iot-greengrass removed
```

Solución de problemas del snap de AWS IoT Greengrass

Utilice la siguiente información como ayuda para solucionar problemas con el snap de AWS IoT Greengrass.

Con errores de permiso denegado.

Solución: los errores de permiso denegado suelen deberse a la falta de interfaces. Para obtener la lista de las interfaces que faltan y la información detallada sobre la solución de problemas, puede utilizar la herramienta `snappy-debug`.

Ejecute el siguiente comando para instalar la herramienta.

```
sudo snap install snappy-debug
```

Respuesta de ejemplo:

```
snappy-debug 0.36-snapd2.45.1 from Canonical# installed
```

Ejecute el comando `sudo snappy-debug` en una nueva sesión de terminal. La operación continúa hasta que se produce un error de permiso denegado.

Por ejemplo, si la función de Lambda intenta leer un archivo del directorio `$HOME`, es posible que obtenga la siguiente respuesta:

```
INFO: Following '/var/log/syslog'. If have dropped messages, use:
INFO: $ sudo journalctl --output=short --follow --all | sudo snappy-debug
kernel.printk_ratelimit = 0
= AppArmor =
Time: Dec  6 04:48:26
Log: apparmor="DENIED" operation="mknod" profile="snap.aws-iot-greengrass.greengrassd"
 name="/home/ubuntu/my-file.txt" pid=12345 comm="touch" requested_mask="c"
 denied_mask="c" fsuid=0 ouid=0
File: /home/ubuntu/my-file.txt (write)
Suggestion:
* add 'home' to 'plugs'
```

En este ejemplo se muestra que la creación del archivo `/home/ubuntu/my-file.txt` provocó el error de permiso. También sugiere que añada home a `plugs`. Sin embargo, esta sugerencia no se puede aplicar. Los enchufes `home-for-greengrassd` y `home-for-hooks` solo tienen acceso de solo lectura.

Para obtener más información, consulte Depuración rápida de snap en la documentación de Snap.

error: no se pueden realizar las siguientes tareas: - Ejecute el comando de servicio «start» para los servicios ["greengrassd"] de snap "" ([start snap. aws-iot-greengrass aws-iot-greengrass.greengrassd.service] falló con el estado de salida 1: Job for snap. aws-iot-greengrass.greengrassd.service falló porque el proceso de control finalizó con un código de error. Consulte «resumen de estado de systemctl». aws-iot-greengrass.greengrassd.service y journalctl -xe para obtener más información.)

Solución: Es posible que vea este error cuando el comando`snap start aws-iot-greengrass` no consigue iniciar el software de AWS IoT Greengrass Core.

Para obtener más información de solución de problemas, ejecute el comando siguiente:

```
sudo snap run aws-iot-greengrass.greengrassd
```

Respuesta de ejemplo:

```
Couldn't find /snap/aws-iot-greengrass/44/greengrass/config/config.json.
```

En este ejemplo, se muestra que AWS IoT Greengrass no pudo encontrar el archivo `config.json`. Puede comprobar los archivos de configuración y de certificado.

/var/snap/ /current/ /packages/1.11.5/rootfs/merged no es una ruta absoluta ni es un enlace aws-iot-greengrass simbólico. ggc-write-directory

Solución: el snap AWS IoT Greengrass solo admite funciones de Lambda no organizadas en contenedores. Asegúrese de que ejecuta su función de Lambda se ejecute como usuario y en modo sin contenedor. Para obtener más información, consulte Consideraciones a la hora de elegir la contenedorización de funciones de Lambda en la Guía del desarrollador de AWS IoT Greengrass Version 1.

El snapd daemon no se pudo reiniciar después de ejecutar el comando sudo snap actualizar snapd.

Solución: siga los pasos 6 a 8 de Instalación y configuración del snap de AWS IoT Greengrass para añadir los archivos de certificado y de configuración de AWS IoT Greengrass al snap AWS IoT Greengrass.

Archivado de una instalación del software AWS IoT Greengrass Core

Cuando actualice a una nueva versión del software de AWS IoT Greengrass Core, puede archivar la versión actualmente instalada. De esta forma, conservará el entorno de instalación actual para que pueda probar una nueva versión de software en el mismo hardware. Esto permite también revertir la versión archivada.

Para archivar la instalación actual e instalar una nueva versión

1. Descargue el paquete de instalación del software AWS IoT Greengrass Core al que desee actualizar.
2. Copie el paquete en el dispositivo de núcleo de destino. Para obtener instrucciones sobre cómo transferir archivos, consulte este paso.

> **Note**
>
> Copiará los certificados, las claves y el archivo de configuración actuales en la nueva instalación más adelante.

Ejecute los comandos de los siguientes pasos en el terminal del dispositivo de núcleo.

3. Asegúrese de que el demonio de Greengrass está detenido en el dispositivo de núcleo.

   a. Para comprobar si el daemon está en ejecución:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, el daemon está en ejecución.

   > **Note**
   >
   > Este procedimiento parte del supuesto de que el software de AWS IoT Greengrass Core está instalado en el directorio `/greengrass`.

   b. Para detener el daemon :

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd stop
   ```

4. Mueva el directorio raíz de Greengrass actual a otro directorio.

   ```
   sudo mv /greengrass /greengrass_backup
   ```

5. Descomprima el nuevo software en el dispositivo de núcleo. Sustituya los marcadores de posición *`os-architecture`* y *`version`* del comando.

   ```
   sudo tar -zxvf greengrass-os-architecture-version.tar.gz -C /
   ```

6. Copie los certificados, las claves y el archivo de configuración archivados en la nueva instalación.

```
sudo cp /greengrass_backup/certs/* /greengrass/certs
sudo cp /greengrass_backup/config/* /greengrass/config
```

7. Inicie el demonio:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

Ahora, puede crear una implementación de grupo para probar la nueva instalación. Si se produce algún error, puede restaurar la instalación archivada.

Para restaurar la instalación archivada

1. Detenga el demonio.
2. Elimine el nuevo directorio `/greengrass`.
3. Mueva el directorio `/greengrass_backup` de nuevo a `/greengrass`.
4. Inicie el daemon.

Configuración de AWS IoT Greengrass Core

Un núcleo AWS IoT Greengrass es una cosa (dispositivo) de AWS IoT que actúa como un concentrador o puerta de enlace en entornos de periferia. Al igual que ocurre con otros dispositivos AWS IoT, hay un núcleo en el registro, tiene una sombra de dispositivo y utiliza un certificado de dispositivo para la autenticación con AWS IoT Core y AWS IoT Greengrass. El dispositivo del núcleo ejecuta el software AWS IoT Greengrass Core, que le permite administrar los procesos locales de los grupos de Greengrass, como la comunicación, la sincronización de instantáneas y el intercambio de tokens.

El software AWS IoT Greengrass Core proporciona las siguientes funcionalidades:

- Implementación y ejecución local de conectores y funciones de Lambda.
- Procesa los flujos de datos de forma local con exportaciones automáticas a la Nube de AWS.
- Mensajes MQTT a través de la red local entre dispositivos, conectores y funciones de Lambda mediante suscripciones administradas.
- Mensajes MQTT entre AWS IoT y dispositivos, conectores y funciones de Lambda mediante suscripciones administradas.

- Conexiones seguras entre los dispositivos y Nube de AWS mediante la autenticación y la autorización de dispositivos.
- Sincronización de sombras locales de dispositivos. Las sombras se pueden configurar para sincronizarse con la Nube de AWS.
- Acceso controlado a los recursos del dispositivo local y el volumen.
- Implementación de modelos de machine learning entrenados en la nube para la ejecución de la inferencia local.
- Detección automática de direcciones IP que permite a los dispositivos detectar el dispositivo principal de Greengrass.
- Implementación central de la configuración de grupos nuevos o actualizados. Una vez descargados los datos de configuración, el dispositivo principal se reinicia automáticamente.
- Actualizaciones de software seguras over-the-air (OTA) de funciones Lambda definidas por el usuario.
- Almacenamiento seguro cifrado de los secretos locales y acceso controlado por conectores y funciones de Lambda.

Archivo de configuración de AWS IoT Greengrass Core

El archivo de configuración del software AWS IoT Greengrass Core es `config.json`. Se encuentra en el directorio `/greengrass-root/config`.

> **Note**
>
> *greengrass-root* representa la ruta donde está instalado el software de AWS IoT Greengrass Core en su dispositivo. Normalmente, este es el directorio `/greengrass`.
> Si utiliza la opción Creación de grupo predeterminada de la consola AWS IoT Greengrass, el archivo `config.json` se implementa en el dispositivo principal en un estado de trabajo.

Para revisar el contenido de este archivo, ejecute el siguiente comando:

```
cat /greengrass-root/config/config.json
```

A continuación se muestra un ejemplo de un archivo `config.json`. Esta es la versión que se genera al crear el núcleo de la consola AWS IoT Greengrass.

GGC v1.11

```
{
    "coreThing": {
        "caPath": "root.ca.pem",
        "certPath": "hash.cert.pem",
        "keyPath": "hash.private.key",
        "thingArn": "arn:partition:iot:region:account-id:thing/core-thing-name",
        "iotHost": "host-prefix-ats.iot.region.amazonaws.com",
        "ggHost": "greengrass-ats.iot.region.amazonaws.com",
        "keepAlive": 600,
        "ggDaemonPort": 8000,
        "systemComponentAuthTimeout": 5000
    },
    "runtime": {
        "maxWorkItemCount": 1024,
        "maxConcurrentLimit": 25,
        "lruSize": 25,
        "mountAllBlockDevices": "no",
        "cgroup": {
            "useSystemd": "yes"
        }
    },
    "managedRespawn": false,
    "crypto": {
        "principals": {
            "SecretsManager": {
                "privateKeyPath": "file:///greengrass/certs/hash.private.key"
            },
            "IoTCertificate": {
                "privateKeyPath": "file:///greengrass/certs/hash.private.key",
                "certificatePath": "file:///greengrass/certs/hash.cert.pem"
            }
        },
        "caPath": "file:///greengrass/certs/root.ca.pem"
    },
    "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
    "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

El archivo `config.json` admite las siguientes propiedades:

coreThing

Campo	Descripción	Notas
`caPath`	La ruta al CA raíz de AWS IoT del directorio */greengrass-root* / `certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`certPath`	La ruta al certificado de dispositivo del núcleo del directorio */greengrass-root*/`certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente.
`keyPath`	La ruta de la clave privada del núcleo del directorio */greengrass-root* / `certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente.
`thingArn`	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que representa el dispositivo de núcleo AWS IoT Greengrass.	Busque el ARN para su núcleo en la consola AWS IoT Greengrass en Núcleos o puede ejecutar el comando de CLI `aws greengrass get-core-definition-version` .
`iotHost`	El punto de enlace de AWS IoT.	Busque el punto de conexión en la consola AWS IoT

Campo	Descripción	Notas
		en Configuración o puede ejecutar el comando de CLI `aws iot describe-endpoint --endpoint-type iot:Data-ATS` . Este comando devuelve el enlace de Amazon Trust Services (ATS). Para obtener más información, consulte la documentación sobre Autenticación del servidor. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.

Campo	Descripción	Notas
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Esto es el punto de enlace de `iotHost` con el prefijo de host reemplazado por greengrass (por ejemplo, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Utilizar el mismo Región de AWS que `iotHost`. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.
`iotMqttPort`	Opcional. El número de puerto que se va a utilizar para la comunicación de MQTT con AWS IoT.	Los valores válidos son `8883` o `443`. El valor predeterminado es `8883`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Campo	Descripción	Notas
`iotHttpPort`	Opcional. El número de puerto que se utiliza para crear las conexiones HTTPS con AWS IoT.	Los valores válidos son `8443` o `443`. El valor predeterminado es `8443`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.
`ggMqttPort`	Opcional. El número de puerto que se va a utilizar para la comunicación MQTT a través de la red local.	Los valores válidos son de `1024` a `65535`. El valor predeterminado es `8883`. Para obtener más información, consulte the section called “Puerto MQTT para la mensajería local”.
`ggHttpPort`	Opcional. El número de puerto que se utiliza para crear las conexiones HTTPS con el servicio de AWS IoT Greengrass.	Los valores válidos son `8443` o `443`. El valor predeterminado es `8443`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.
`keepAlive`	Opcional. El periodo de MQTT `KeepAlive` en segundos.	El intervalo válido está comprendido entre 30 y 1200 segundos. El valor predeterminado es `600`.
`networkProxy`	Opcional. Un objeto que define un servidor proxy al que se conectará.	El servidor proxy puede ser HTTP o HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Campo	Descripción	Notas
`mqttOperationTimeout`	Opcional. Cantidad de tiempo (en segundos) para permitir que el núcleo de Greengras s complete una operación de publicación, suscripción o cancelación de suscripción en las conexiones MQTT con AWS IoT Core.	El valor predeterminado es 5. El valor mínimo es 5.
`ggDaemonPort`	Opcional. El número de puerto IPC principal de Greengrass.	Esta propiedad está disponibl e en la versión AWS IoT Greengrass 1.11.0 o posterior. Los valores válidos están comprendidos entre 1024 y 65535. El valor predeterm inado es 8000.
`systemComponentAut hTimeout`	Opcional. Tiempo (en milisegundos) para permitir que el IPC principal de Greengrass complete la autenticación.	Esta propiedad está disponibl e en la versión AWS IoT Greengrass 1.11.0 o posterior. Los valores válidos están comprendidos entre 500 y 5000. El valor predeterm inado es 5000.

runtime

Campo	Descripción	Notas
`maxWorkItemCuente`	Opcional. El número máximo de elementos de trabajo que	El valor predeterminado es 1024. El valor máximo está

Campo	Descripción	Notas
	el daemon de Greengras s puede procesar a la vez. Se ignorarán los elementos de trabajo que superen ese límite. Los componentes del sistema, las funciones de Lambda definidas por el usuario y los conectore s comparten la cola de elementos de trabajo.	limitado por el hardware del dispositivo. Aumentar este valor aumenta la memoria que AWS IoT Greengrass usa. Puede aumentar este valor si espera que su núcleo reciba un gran tráfico de mensajes MQTT.
`maxConcurrentLimit`	Opcional. Número máximo de trabajadores Lambda simultáneos sin fijar que puede tener el daemon Greengrass. Puede especific ar un entero diferente para anular este parámetro.	El valor predeterminado es 25. El valor mínimo está definido por `lruSize`.
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.

Campo	Descripción	Notas
`mountAllBlockDispositivos`	Optional. Enables AWS IoT Greengrass to use bind mounts to mount all block devices into a container after setting up the OverlayFS.	Esta propiedad está disponible en la versión AWS IoT Greengrass 1.11.0 o posterior. Los valores válidos son `yes` y `no`. El valor predeterminado es `no`. Establezca este valor en `yes` si su directorio `/usr` no está por debajo de la jerarquía `/`.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `no`. Run the `check_ggc_dependencies` script in Módulo 1 to see if your device uses `systemd`.

crypto

El `crypto` las propiedades que admiten el almacenamiento de claves privadas en un módulo de seguridad de hardware (HSM) a través de PKCS#11 y almacenamiento secreto local. Para obtener más información, consulte the section called “Entidades de seguridad”, the section called “Integración de la seguridad de hardware” y Implementación de secretos en el núcleo. Se admiten las configuraciones de almacenamiento de claves privadas en HSM o en el sistema de archivos.

Campo	Descripción	Notas
`caPath`	La ruta absoluta al CA raíz de AWS IoT.	Debe ser el URI de un archivo de la forma: `file:///absolute/path/to/file` . **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. La ruta absoluta al archivo `.so` del motor de OpenSSL para habilitar la compatibilidad con PKCS#11 en OpenSSL.	Debe ser una ruta a un archivo del sistema de archivos. Esta propiedad es necesaria si está utilizando el agente de actualización OTA de Greengrass con seguridad de hardware. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.
`Proveedor de P11`	La ruta absoluta a la bibliotec a que puede cargar libdl de la implementación de PKCS#11.	Debe ser una ruta a un archivo del sistema de archivos.
`slotLabel`	La etiqueta de ranura que se utiliza para identificar el módulo de hardware.	Debe ajustarse a las especificaciones de etiqueta de PKCS#11.

Campo	Descripción	Notas
`slotUserPin`	El PIN de usuario que se utiliza para autenticar el núcleo de Greengrass en el módulo.	Debe tener permisos suficientes para realizar la firma C_Sign con las claves privadas configuradas.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`Certificado IoT. privateKeyPath`	La ruta a la clave privada del núcleo.	Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto.
`IoTCertificate .certificatePath`	La ruta absoluta al certificado de dispositivo del núcleo.	Debe ser el URI de un archivo de la forma: `file:///`*`absolute/path/to/file`* .
`MQTT ServerCertificate`	Opcional. La clave privada que el núcleo utiliza en combinación con el certificado para que actúe como un servidor MQTT o la puerta de enlace.	

Campo	Descripción	Notas
`MQTTServerCertificate . privateKeyPath`	La ruta a la clave privada del servidor MQTT local.	Utilice este valor para especificar su propia clave privada para el servidor MQTT local. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. Si esta propiedad se omite, AWS IoT Greengrass rota la clave en función de la configuración de rotación. Si se especifica, el cliente es responsable de la rotación de la clave.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Implementación de secretos en el núcleo.	

Campo	Descripción	Notas
`SecretsManager.privateKeyPath`	La ruta a la clave privada del administrador de secretos locales.	Solo se admite una clave RSA. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///absolute/path/to/file` . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. La clave privada se debe generar utilizando el mecanismo de rellenado PKCS#1 v1.5.

También se admiten las siguientes propiedades de configuración:

Campo	Descripción	Notas
`mqttMaxConnectionRetryInterval`	Opcional. El intervalo máximo (en segundos) entre los reintentos de conexión MQTT si se pierde la conexión.	Especifique este valor como un número entero sin firmar. El valor predeterminado es `60`.
`managedRespawn`	Opcional. Indica que el agente de OTA debe ejecutar un código personalizado antes de una actualización.	Los valores válidos son `true` o `false`. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.

Campo	Descripción	Notas
`writeDirectory`	Opcional. El directorio de escritura donde AWS IoT Greengrass crea todos los recursos de lectura y escritura.	Para obtener más información, consulte Configurar un directorio de escritura para AWS IoT Greengrass.
pidFileDirectory	Opcional. AWS IoT Greengrass almacena su ID de proceso (PID) en este directorio.	El valor predeterminado es `/var/run`.

Extended life versions

Las siguientes versiones del software de AWS IoT Greengrass Core se encuentran en la fase de vida útil prolongada. Esta información se incluye únicamente con fines de referencia.

GGC v1.10

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600,
    "systemComponentAuthTimeout": 5000
  },
  "runtime" : {
    "maxWorkItemCount" : 1024,
    "maxConcurrentLimit" : 25,
    "lruSize": 25,
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
```

```
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

El archivo `config.json` admite las siguientes propiedades:

coreThing

Campo	Descripción	Notas
`caPath`	La ruta al CA raíz de AWS IoT del directorio */greengrass-root* / `certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`certPath`	La ruta al certificado de dispositivo del núcleo del	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta

Campo	Descripción	Notas
	directorio */greengrass-root*/certs.	propiedad se ignora cuando el objeto crypto está presente.
keyPath	La ruta de la clave privada del núcleo del directorio */greengrass-root* /certs.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto crypto está presente.
thingArn	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que representa el dispositivo de núcleo AWS IoT Greengrass.	Busque el ARN para su núcleo en la consola AWS IoT Greengrass en Núcleos o puede ejecutar el comando de CLI aws greengrass get-core-definition-version.

Campo	Descripción	Notas
`iotHost`	El punto de enlace de AWS IoT.	Busque el punto de conexión en la consola AWS IoT en Configuración o puede ejecutar el comando de CLI `aws iot describe-endpoint --endpoint-type iot:Data-ATS`. Este comando devuelve el enlace de Amazon Trust Services (ATS). Para obtener más información, consulte la documentación sobre Autenticación del servidor. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.

Campo	Descripción	Notas
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Esto es el punto de enlace de `iotHost` con el prefijo de host reemplazado por greengrass (por ejemplo, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Utilizar el mismo Región de AWS que `iotHost`. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.
`iotMqttPort`	Opcional. El número de puerto que se va a utilizar para la comunicación de MQTT con AWS IoT.	Los valores válidos son `8883` o `443`. El valor predeterminado es `8883`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Campo	Descripción	Notas
`iotHttpPort`	Opcional. El número de puerto que se utiliza para crear las conexiones HTTPS con AWS IoT.	Los valores válidos son `8443` o `443`. El valor predeterminado es `8443`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.
`ggMqttPort`	Opcional. El número de puerto que se va a utilizar para la comunicación MQTT a través de la red local.	Los valores válidos son de `1024` a `65535`. El valor predeterminado es `8883`. Para obtener más información, consulte the section called “Puerto MQTT para la mensajería local”.
`ggHttpPort`	Opcional. El número de puerto que se utiliza para crear las conexiones HTTPS con el servicio de AWS IoT Greengrass.	Los valores válidos son `8443` o `443`. El valor predeterminado es `8443`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.
`keepAlive`	Opcional. El periodo de MQTT `KeepAlive` en segundos.	El intervalo válido está comprendido entre 30 y 1200 segundos. El valor predeterminado es `600`.
`networkProxy`	Opcional. Un objeto que define un servidor proxy al que se conectará.	El servidor proxy puede ser HTTP o HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Campo	Descripción	Notas
`mqttOperationTimeout`	Opcional. Cantidad de tiempo (en segundos) para permitir que el núcleo de Greengrass complete una operación de publicación, suscripción o cancelación de suscripción en las conexiones MQTT con AWS IoT Core.	Esta propiedad está disponible a partir de AWS IoT Greengrass v1.10.2. El valor predeterminado es 5. El valor mínimo es 5.

runtime

Campo	Descripción	Notas
`maxWorkItemContar`	Opcional. El número máximo de elementos de trabajo que el daemon de Greengrass puede procesar a la vez. Se ignorarán los elementos de trabajo que superen ese límite. Los componentes del sistema, las funciones de Lambda definidas por el usuario y los conectores comparten la cola de elementos de trabajo.	El valor predeterminado es 1024. El valor máximo está limitado por el hardware del dispositivo. Aumentar este valor aumenta la memoria que AWS IoT Greengrass usa. Puede aumentar este valor si espera que su núcleo reciba un gran tráfico de mensajes MQTT.
`maxConcurrentLimit`	Opcional. Número máximo de trabajadores Lambda simultáneos sin fijar que puede tener el daemon Greengrass. Puede especificar un entero	El valor predeterminado es 25. El valor mínimo está definido por `lruSize`.

Campo	Descripción	Notas
	diferente para anular este parámetro.	
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `no`. Run the `check_ggc_dependencies` script in Módulo 1 to see if your device uses `systemd`.

crypto

El `crypto` las propiedades que admiten el almacenamiento de claves privadas en un módulo de seguridad de hardware (HSM) a través de PKCS#11 y almacenamiento secreto local. Para obtener más información, consulte the section called “Entidades de seguridad”, the section called “Integración de la seguridad de hardware” y Implementación de secretos en el núcleo. Se admiten las configuraciones de almacenamiento de claves privadas en HSM o en el sistema de archivos.

Campo	Descripción	Notas
`caPath`	La ruta absoluta al CA raíz de AWS IoT.	Debe ser el URI de un archivo de la forma: `file:///absolute/path/to/file` .

Campo	Descripción	Notas
		Note Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. La ruta absoluta al archivo `.so` del motor de OpenSSL para habilitar la compatibilidad con PKCS#11 en OpenSSL.	Debe ser una ruta a un archivo del sistema de archivos. Esta propiedad es necesaria si está utilizando el agente de actualización OTA de Greengrass con seguridad de hardware. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.
`Proveedor de P11`	La ruta absoluta a la biblioteca que puede cargar libdl de la implementación de PKCS#11.	Debe ser una ruta a un archivo del sistema de archivos.
`slotLabel`	La etiqueta de ranura que se utiliza para identificar el módulo de hardware.	Debe ajustarse a las especificaciones de etiqueta de PKCS#11.

Campo	Descripción	Notas
`slotUserPin`	El PIN de usuario que se utiliza para autenticar el núcleo de Greengrass en el módulo.	Debe tener permisos suficientes para realizar la firma C_Sign con las claves privadas configuradas.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`Certificado de IoT. privateKeyPath`	La ruta a la clave privada del núcleo.	Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto.
`IoTCertificate .certificatePath`	La ruta absoluta al certificado de dispositivo del núcleo.	Debe ser el URI de un archivo de la forma: `file:///`*`absolute/path/to/file`* .
`MQTT ServerCertificate`	Opcional. La clave privada que el núcleo utiliza en combinación con el certificado para que actúe como un servidor MQTT o la puerta de enlace.	

Campo	Descripción	Notas
`MQTTServerCertificate . privateKeyPath`	La ruta a la clave privada del servidor MQTT local.	Utilice este valor para especificar su propia clave privada para el servidor MQTT local. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. Si esta propiedad se omite, AWS IoT Greengrass rota la clave en función de la configuración de rotación. Si se especifica, el cliente es responsable de la rotación de la clave.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Implementación de secretos en el núcleo.	

Campo	Descripción	Notas
`SecretsManager .privateKeyPath`	La ruta a la clave privada del administrador de secretos locales.	Solo se admite una clave RSA. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. La clave privada se debe generar utilizando el mecanismo de rellenado PKCS#1 v1.5.

También se admiten las siguientes propiedades de configuración:

Campo	Descripción	Notas
`mqttMaxConnectionRetryInterval`	Opcional. El intervalo máximo (en segundos) entre los reintentos de conexión MQTT si se pierde la conexión.	Especifique este valor como un número entero sin firmar. El valor predeterminado es `60`.
`managedRespawn`	Opcional. Indica que el agente de OTA debe ejecutar un código personalizado antes de una actualización.	Los valores válidos son `true` o `false`. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.

Campo	Descripción	Notas
`writeDirectory`	Opcional. El directorio de escritura donde AWS IoT Greengrass crea todos los recursos de lectura y escritura.	Para obtener más información, consulte Configurar un directorio de escritura para AWS IoT Greengrass.

GGC v1.9

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

El archivo `config.json` admite las siguientes propiedades:

coreThing

Campo	Descripción	Notas
`caPath`	La ruta al CA raíz de AWS IoT del directorio */greengrass-root* /`certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`certPath`	La ruta al certificado de dispositivo del núcleo del directorio */greengrass-root*/`certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente.
`keyPath`	La ruta de la clave privada del núcleo del directorio */greengrass-root* /`certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente.
`thingArn`	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que represent	Busque el ARN para su núcleo en la consola AWS IoT Greengrass en

Campo	Descripción	Notas
	a el dispositivo de núcleo AWS IoT Greengrass.	Núcleos o puede ejecutar el comando de CLI `aws greengrass get-core-definition-version`.

Campo	Descripción	Notas
`iotHost`	El punto de enlace de AWS IoT.	Busque el punto de conexión en la consola AWS IoT en Configuración o puede ejecutar el comando de CLI `aws iot describe-endpoint --endpoint-type iot:Data-ATS`. Este comando devuelve el enlace de Amazon Trust Services (ATS). Para obtener más información, consulte la documentación sobre Autenticación del servidor. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.

Campo	Descripción	Notas
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Esto es el punto de enlace de `iotHost` con el prefijo de host reemplazado por greengrass (por ejemplo, `greengrass-ats.iot.region.amazonaws.com`). Utilizar el mismo Región de AWS que `iotHost`. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.
`iotMqttPort`	Opcional. El número de puerto que se va a utilizar para la comunicación de MQTT con AWS IoT.	Los valores válidos son `8883` o `443`. El valor predeterminado es `8883`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Campo	Descripción	Notas
`iotHttpPort`	Opcional. El número de puerto que se utiliza para crear las conexiones HTTPS con AWS IoT.	Los valores válidos son `8443` o `443`. El valor predeterminado es `8443`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.
`ggHttpPort`	Opcional. El número de puerto que se utiliza para crear las conexiones HTTPS con el servicio de AWS IoT Greengrass.	Los valores válidos son `8443` o `443`. El valor predeterminado es `8443`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.
`keepAlive`	Opcional. El periodo de MQTT `KeepAlive` en segundos.	El intervalo válido está comprendido entre 30 y 1200 segundos. El valor predeterminado es `600`.
`networkProxy`	Opcional. Un objeto que define un servidor proxy al que se conectará.	El servidor proxy puede ser HTTP o HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

runtime

Campo	Descripción	Notas
`maxConcurrentLimit`	Opcional. Número máximo de trabajadores Lambda simultáneos sin fijar que	El valor predeterminado es 25. El valor mínimo está definido por `lruSize`.

Campo	Descripción	Notas
	puede tener el daemon Greengrass. Puede especificar un entero diferente para anular este parámetro.	
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `no`. Run the `check_ggc_dependencies` script in Módulo 1 to see if your device uses `systemd`.

crypto

El objeto `crypto` se añade en v1.7.0. Introduce las propiedades que admiten el almacenamiento de claves privadas en un módulo de seguridad de hardware (HSM) a través de PKCS#11 y almacenamiento secreto local. Para obtener más información, consulte the section called “Entidades de seguridad”, the section called “Integración de la seguridad de hardware” y Implementación de secretos en el núcleo. Se admiten las configuraciones de almacenamiento de claves privadas en HSM o en el sistema de archivos.

Campo	Descripción	Notas
`caPath`	La ruta absoluta al CA raíz de AWS IoT.	Debe ser el URI de un archivo de la forma: `file:///`*`absolute/path/to/file`* . **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. La ruta absoluta al archivo `.so` del motor de OpenSSL para habilitar la compatibilidad con PKCS#11 en OpenSSL.	Debe ser una ruta a un archivo del sistema de archivos. Esta propiedad es necesaria si está utilizando el agente de actualización OTA de Greengrass con seguridad de hardware. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.
`Proveedor de P11`	La ruta absoluta a la biblioteca que puede cargar libdl de la implementación de PKCS#11.	Debe ser una ruta a un archivo del sistema de archivos.

Campo	Descripción	Notas
`slotLabel`	La etiqueta de ranura que se utiliza para identificar el módulo de hardware.	Debe ajustarse a las especificaciones de etiqueta de PKCS#11.
`slotUserPin`	El PIN de usuario que se utiliza para autenticar el núcleo de Greengrass en el módulo.	Debe tener permisos suficientes para realizar la firma C_Sign con las claves privadas configuradas.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`Certificado IoT. privateKeyPath`	La ruta a la clave privada del núcleo.	Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto.
`IoTCertificate .certificatePath`	La ruta absoluta al certificado de dispositivo del núcleo.	Debe ser el URI de un archivo de la forma: `file:///`*`absolute/path/to/file`* .
`MQTT ServerCertificate`	Opcional. La clave privada que el núcleo utiliza en combinación con el certificado para que actúe como un servidor MQTT o la puerta de enlace.	

Campo	Descripción	Notas
`MQTTServerCertificate . privateKeyPath`	La ruta a la clave privada del servidor MQTT local.	Utilice este valor para especificar su propia clave privada para el servidor MQTT local. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. Si esta propiedad se omite, AWS IoT Greengrass rota la clave en función de la configuración de rotación. Si se especifica, el cliente es responsable de la rotación de la clave.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Implementación de secretos en el núcleo.	

Campo	Descripción	Notas
`SecretsManager .privateKeyPath`	La ruta a la clave privada del administrador de secretos locales.	Solo se admite una clave RSA. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. La clave privada se debe generar utilizando el mecanismo de rellenado PKCS#1 v1.5.

También se admiten las siguientes propiedades de configuración.

Campo	Descripción	Notas
`mqttMaxConnectionRetryInterval`	Opcional. El intervalo máximo (en segundos) entre los reintentos de conexión MQTT si se pierde la conexión.	Especifique este valor como un número entero sin firmar. El valor predeterminado es `60`.
`managedRespawn`	Opcional. Indica que el agente de OTA debe ejecutar un código personalizado antes de una actualización.	Los valores válidos son `true` o `false`. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.

Campo	Descripción	Notas
`writeDirectory`	Opcional. El directorio de escritura donde AWS IoT Greengrass crea todos los recursos de lectura y escritura.	Para obtener más información, consulte Configurar un directorio de escritura para AWS IoT Greengrass.

GGC v1.8

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

El archivo `config.json` admite las siguientes propiedades.

coreThing

Campo	Descripción	Notas
`caPath`	La ruta al CA raíz de AWS IoT del directorio */greengrass-root* / `certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`certPath`	La ruta al certificado de dispositivo del núcleo del directorio */greengrass-root*/`certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente.
`keyPath`	La ruta de la clave privada del núcleo del directorio */greengrass-root* / `certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente.
`thingArn`	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que represent	Busque el ARN para su núcleo en la consola AWS IoT Greengrass en

Campo	Descripción	Notas
	a el dispositivo de núcleo AWS IoT Greengrass.	Núcleos o puede ejecutar el comando de CLI `aws greengrass get-core-definition-version` .
`iotHost`	El punto de enlace de AWS IoT.	Busque el punto de conexión en la consola AWS IoT en Configura ción o puede ejecutar el comando de CLI `aws iot describe-endpoint --endpoint-type iot:Data-ATS` . Este comando devuelve el enlace de Amazon Trust Services (ATS). Para obtener más información, consulte la documentación sobre Autenticación del servidor. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certifica do. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.

Campo	Descripción	Notas
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Esto es el punto de enlace de `iotHost` con el prefijo de host reemplazado por greengrass (por ejemplo, `greengrass-ats.iot.region.amazonaws.com`). Utilizar el mismo Región de AWS que `iotHost`. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.
`iotMqttPort`	Opcional. El número de puerto que se va a utilizar para la comunicación de MQTT con AWS IoT.	Los valores válidos son `8883` o `443`. El valor predeterminado es `8883`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Campo	Descripción	Notas
`iotHttpPort`	Opcional. El número de puerto que se utiliza para crear las conexiones HTTPS con AWS IoT.	Los valores válidos son `8443` o `443`. El valor predeterminado es `8443`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.
`ggHttpPort`	Opcional. El número de puerto que se utiliza para crear las conexiones HTTPS con el servicio de AWS IoT Greengrass.	Los valores válidos son `8443` o `443`. El valor predeterminado es `8443`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.
`keepAlive`	Opcional. El periodo de MQTT `KeepAlive` en segundos.	El intervalo válido está comprendido entre 30 y 1200 segundos. El valor predeterminado es `600`.
`networkProxy`	Opcional. Un objeto que define un servidor proxy al que se conectará.	El servidor proxy puede ser HTTP o HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

runtime

Campo	Descripción	Notas
`cgroup`		

Campo	Descripción	Notas
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `no`. Run the `check_ggc_dependencies` script in Módulo 1 to see if your device uses `systemd`.

crypto

El objeto `crypto` se añade en v1.7.0. Introduce las propiedades que admiten el almacenamiento de claves privadas en un módulo de seguridad de hardware (HSM) a través de PKCS#11 y almacenamiento secreto local. Para obtener más información, consulte the section called “Entidades de seguridad”, the section called “Integración de la seguridad de hardware” y Implementación de secretos en el núcleo. Se admiten las configuraciones de almacenamiento de claves privadas en HSM o en el sistema de archivos.

Campo	Descripción	Notas
`caPath`	La ruta absoluta al CA raíz de AWS IoT.	Debe ser el URI de un archivo de la forma: `file:///absolute/path/to/file` . **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.

`PKCS11`

Campo	Descripción	Notas
`OpenSSLEngine`	Opcional. La ruta absoluta al archivo `.so` del motor de OpenSSL para habilitar la compatibilidad con PKCS#11 en OpenSSL.	Debe ser una ruta a un archivo del sistema de archivos. Esta propiedad es necesaria si está utilizando el agente de actualización OTA de Greengrass con seguridad de hardware. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.
`Proveedor de P11`	La ruta absoluta a la biblioteca que puede cargar libdl de la implementación de PKCS#11.	Debe ser una ruta a un archivo del sistema de archivos.
`slotLabel`	La etiqueta de ranura que se utiliza para identificar el módulo de hardware.	Debe ajustarse a las especificaciones de etiqueta de PKCS#11.
`slotUserPin`	El PIN de usuario que se utiliza para autenticar el núcleo de Greengrass en el módulo.	Debe tener permisos suficientes para realizar la firma C_Sign con las claves privadas configuradas.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	

Campo	Descripción	Notas
`Certificado IoT. privateKeyPath`	La ruta a la clave privada del núcleo.	Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto.
`IoTCertificate .certificatePath`	La ruta absoluta al certificado de dispositivo del núcleo.	Debe ser el URI de un archivo de la forma: `file:///`*`absolute/path/to/file`* .
`MQTT ServerCertificate`	Opcional. La clave privada que el núcleo utiliza en combinación con el certificado para que actúe como un servidor MQTT o la puerta de enlace.	

Campo	Descripción	Notas
`MQTTServerCertificate . privateKeyPath`	La ruta a la clave privada del servidor MQTT local.	Utilice este valor para especificar su propia clave privada para el servidor MQTT local. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. Si esta propiedad se omite, AWS IoT Greengrass rota la clave en función de la configuración de rotación. Si se especifica, el cliente es responsable de la rotación de la clave.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Implementación de secretos en el núcleo.	

Campo	Descripción	Notas
`SecretsManager .privateKeyPath`	La ruta a la clave privada del administrador de secretos locales.	Solo se admite una clave RSA. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. La clave privada se debe generar utilizando el mecanismo de rellenado PKCS#1 v1.5.

También se admiten las siguientes propiedades de configuración:

Campo	Descripción	Notas
`mqttMaxConnectionRetryInterval`	Opcional. El intervalo máximo (en segundos) entre los reintentos de conexión MQTT si se pierde la conexión.	Especifique este valor como un número entero sin firmar. El valor predeterminado es `60`.
`managedRespawn`	Opcional. Indica que el agente de OTA debe ejecutar un código personalizado antes de una actualización.	Los valores válidos son `true` o `false`. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.

Campo	Descripción	Notas
`writeDirectory`	Opcional. El directorio de escritura donde AWS IoT Greengrass crea todos los recursos de lectura y escritura.	Para obtener más información, consulte Configurar un directorio de escritura para AWS IoT Greengrass.

GGC v1.7

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

El archivo `config.json` admite las siguientes propiedades:

coreThing

Campo	Descripción	Notas
`caPath`	La ruta al CA raíz de AWS IoT del directorio */greengrass-root* /`certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`certPath`	La ruta al certificado de dispositivo del núcleo del directorio */greengrass-root*/`certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente.
`keyPath`	La ruta de la clave privada del núcleo del directorio */greengrass-root* /`certs`.	Para garantizar la compatibilidad con versiones anteriores a la 1.7.0. Esta propiedad se ignora cuando el objeto `crypto` está presente.
`thingArn`	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que represent	Busque el ARN para su núcleo en la consola AWS IoT Greengrass en

Campo	Descripción	Notas
	a el dispositivo de núcleo AWS IoT Greengrass.	Núcleos o puede ejecutar el comando de CLI `aws greengrass get-core-definition-version` .
`iotHost`	El punto de enlace de AWS IoT.	Busque el punto de conexión en la consola AWS IoT en Configura ción o puede ejecutar el comando de CLI `aws iot describe-endpoint --endpoint-type iot:Data-ATS` . Este comando devuelve el enlace de Amazon Trust Services (ATS). Para obtener más información, consulte la documentación sobre Autenticación del servidor. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certifica do. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.

Campo	Descripción	Notas
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Esto es el punto de enlace de `iotHost` con el prefijo de host reemplazado por greengrass (por ejemplo, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Utilizar el mismo Región de AWS que `iotHost`. **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado. Asegúrese de que los puntos de conexión se corresponden con su Región de AWS.
`iotMqttPort`	Opcional. El número de puerto que se va a utilizar para la comunicación de MQTT con AWS IoT.	Los valores válidos son `8883` o `443`. El valor predeterminado es `8883`. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Campo	Descripción	Notas
`keepAlive`	Opcional. El periodo de MQTT `KeepAlive` en segundos.	El intervalo válido está comprendido entre 30 y 1200 segundos. El valor predeterminado es `600`.
`networkProxy`	Opcional. Un objeto que define un servidor proxy al que se conectará.	El servidor proxy puede ser HTTP o HTTPS. Para obtener más informaci ón, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

runtime

Campo	Descripción	Notas
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `no`. Run the `check_ggc _dependencies` script in Módulo 1 to see if your device uses `systemd`.

crypto

El objeto `crypto`, añadido en la v1.7.0, introduce las propiedades que admiten el almacenamiento de claves privadas en un módulo de seguridad de hardware (HSM) a través de PKCS#11 y almacenamiento secreto local. Para obtener más información, consulte the section called “Integración de la seguridad de hardware” y Implementación de secretos en el núcleo. Se admiten las configuraciones de almacenamiento de claves privadas en HSM o en el sistema de archivos.

Campo	Descripción	Notas
`caPath`	La ruta absoluta al CA raíz de AWS IoT.	Debe ser el URI de un archivo de la forma: `file:///`*`absolute/path/to/file`* . **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. La ruta absoluta al archivo `.so` del motor de OpenSSL para habilitar la compatibilidad con PKCS#11 en OpenSSL.	Debe ser una ruta a un archivo del sistema de archivos. Esta propiedad es necesaria si está utilizando el agente de actualización OTA de Greengrass con seguridad de hardware. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.
`Proveedor de P11`	La ruta absoluta a la biblioteca que puede cargar libdl de la implementación de PKCS#11.	Debe ser una ruta a un archivo del sistema de archivos.

Campo	Descripción	Notas
`slotLabel`	La etiqueta de ranura que se utiliza para identificar el módulo de hardware.	Debe ajustarse a las especificaciones de etiqueta de PKCS#11.
`slotUserPin`	El PIN de usuario que se utiliza para autenticar el núcleo de Greengrass en el módulo.	Debe tener permisos suficientes para realizar la firma C_Sign con las claves privadas configuradas.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`Certificado IoT. privateKeyPath`	La ruta a la clave privada del núcleo.	Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto.
`IoTCertificate .certificatePath`	La ruta absoluta al certificado de dispositivo del núcleo.	Debe ser el URI de un archivo de la forma: `file:///`*`absolute/path/to/file`* .
`MQTT ServerCertificate`	Opcional. La clave privada que el núcleo utiliza en combinación con el certificado para que actúe como un servidor MQTT o la puerta de enlace.	

Campo	Descripción	Notas
`MQTTServerCertificate . privateKeyPath`	La ruta a la clave privada del servidor MQTT local.	Utilice este valor para especificar su propia clave privada para el servidor MQTT local. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. Si esta propiedad se omite, AWS IoT Greengrass rota la clave en función de la configuración de rotación. Si se especifica, el cliente es responsable de la rotación de la clave.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Implementación de secretos en el núcleo.	

Campo	Descripción	Notas
`SecretsManager .privateKeyPath`	La ruta a la clave privada del administrador de secretos locales.	Solo se admite una clave RSA. Para el almacenamiento del sistema de archivos, debe ser un URI de archivo de la forma: `file:///`*`absolute/path/to/file`* . Para el almacenamiento de HSM, debe ser una ruta RFC 7512 PKCS#11 que especifica la etiqueta del objeto. La clave privada se debe generar utilizando el mecanismo de rellenado PKCS#1 v1.5.

También se admiten las siguientes propiedades de configuración:

Campo	Descripción	Notas
`mqttMaxConnectionRetryInterval`	Opcional. El intervalo máximo (en segundos) entre los reintentos de conexión MQTT si se pierde la conexión.	Especifique este valor como un número entero sin firmar. El valor predeterminado es `60`.
`managedRespawn`	Opcional. Indica que el agente de OTA debe ejecutar un código personalizado antes de una actualización.	Los valores válidos son `true` o `false`. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.

Campo	Descripción	Notas
`writeDirectory`	Opcional. El directorio de escritura donde AWS IoT Greengrass crea todos los recursos de lectura y escritura.	Para obtener más información, consulte Configurar un directorio de escritura para AWS IoT Greengrass.

GGC v1.6

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600,
       "mqttMaxConnectionRetryInterval": 60
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   },
   "managedRespawn": true,
   "writeDirectory": "/write-directory"
}
```

> **Note**
>
> Si utiliza la opción Creación de grupo predeterminada de la consola AWS IoT Greengrass, el archivo `config.json` se implementa en el dispositivo principal en un estado de trabajo que especifica la configuración predeterminada.

El archivo `config.json` admite las siguientes propiedades:

Campo	Descripción	Notas
`caPath`	La ruta a la CA raíz de AWS IoT relativa al directorio */greengrass-root* /certs.	Guarde el archivo en */greengrass-root* /certs.
`certPath`	La ruta del certificado de AWS IoT Greengrass del núcleo del directorio */greengrass-root* /certs.	Guarde el archivo en */greengrass-root* /certs.
`keyPath`	La ruta de la clave privada AWS IoT Greengrass del núcleo del directorio */greengrass-root* /certs.	Guarde el archivo en */greengrass-root* /certs.
`thingArn`	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que represent a el dispositivo de núcleo AWS IoT Greengrass.	Busque el ARN para su núcleo en la consola AWS IoT Greengrass en Núcleos o puede ejecutar el comando de CLI `aws greengrass get-core-definition-version` .
`iotHost`	El punto de enlace de AWS IoT.	Busque esto en la consola AWS IoT en Configura ción o puede ejecutar el comando de la CLI `aws iot describe-endpoint` .
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Este valor usa el formato `greengras s.iot.` *region*`.amazonaw`

Campo	Descripción	Notas
		`s.com` . Usa la misma región que `iotHost`.
`keepAlive`	El periodo de MQTT `KeepAlive` en segundos.	Se trata de un valor opcional. El valor predeterminado es `600`.
`mqttMaxConnectionRetryInterval`	El intervalo máximo (en segundos) entre los reintentos de conexión MQTT si se pierde la conexión.	Especifique este valor como un número entero sin firmar. Se trata de un valor opcional. El valor predeterminado es `60`.
`useSystemd`	Indica si su dispositivo usa `systemd`.	Los valores válidos son `yes` o no. Ejecute el script `check_ggc_dependencies` en el Módulo 1 para comprobar si el dispositivo usa `systemd`.
`managedRespawn`	Una función de actualizaciones opcional over-the-air (OTA), que indica que el agente OTA debe ejecutar un código personalizado antes de una actualización.	Los valores válidos son `true` o `false`. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.
`writeDirectory`	El directorio de escritura donde AWS IoT Greengrass crea todos los recursos de lectura y escritura.	Se trata de un valor opcional. Para obtener más información, consulte Configurar un directorio de escritura para AWS IoT Greengrass.

GGC v1.5

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   },
   "managedRespawn": true
}
```

El archivo `config.json` existe en */greengrass-root*`/config` y contiene los parámetros siguientes:

Campo	Descripción	Notas
`caPath`	La ruta a la CA raíz de AWS IoT relativa a la carpeta */greengrass-root* `/ certs`.	Guarde el archivo en la carpeta */greengrass-root*`/certs`.
`certPath`	La ruta del certificado de AWS IoT Greengrass del núcelo de la carpeta */greengrass-root* `/ certs`.	Guarde el archivo en la carpeta */greengrass-root*`/certs`.
`keyPath`	La ruta de la clave privada de AWS IoT Greengrass del núcleo de la carpeta	Guarde el archivo en la carpeta */greengrass-root*`/certs`.

Campo	Descripción	Notas
	`/greengrass-root /certs`.	
`thingArn`	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que represent a el dispositivo de núcleo AWS IoT Greengrass.	Busque el ARN para su núcleo en la consola AWS IoT Greengrass en Núcleos o puede ejecutar el comando de CLI `aws greengrass get-core-definition-version`.
`iotHost`	El punto de enlace de AWS IoT.	Busque esto en la consola AWS IoT en Configuración o puede ejecutar el comando `aws iot describe-endpoint`.
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Este valor usa el formato `greengras s.iot. region.amazonaw s.com`. Usa la misma región que `iotHost`.
`keepAlive`	El periodo de MQTT `KeepAlive` en segundos.	Se trata de un valor opcional. El valor predeterm inado es 600 segundos.
`useSystemd`	Indica si su dispositivo usa `systemd`.	Los valores válidos son `yes` o no. Ejecute el script `check_ggc_dependen cies` en el Módulo 1 para comprobar si el dispositivo usa `systemd`.

Campo	Descripción	Notas
`managedRespawn`	Una función de actualizaciones opcional over-the-air (OTA) que indica que el agente de OTA debe ejecutar un código personalizado antes de realizar una actualización.	Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.

GGC v1.3

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   },
   "managedRespawn": true
}
```

El archivo `config.json` existe en */greengrass-root*`/config` y contiene los parámetros siguientes:

Campo	Descripción	Notas
`caPath`	La ruta a la CA raíz de AWS IoT relativa a la carpeta	Guarde el archivo en la carpeta */greengrass-root*`/certs`.

Campo	Descripción	Notas
	/greengrass-root /certs.	
certPath	La ruta del certificado de AWS IoT Greengrass del núcelo de la carpeta */greengrass-root* /certs.	Guarde el archivo en la carpeta */greengrass-root*/certs.
keyPath	La ruta de la clave privada de AWS IoT Greengrass del núcleo de la carpeta */greengrass-root* /certs.	Guarde el archivo en la carpeta */greengrass-root*/certs.
thingArn	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que representa el AWS IoT Greengrass del núcleo.	Este valor se puede encontrar en la consola AWS IoT Greengrass, en la definición de su objeto de AWS IoT.
iotHost	El punto de enlace de AWS IoT.	Este valor se puede encontrar en la consola AWS IoT, en Configuración.
ggHost	El punto de enlace de AWS IoT Greengrass.	Este valor se puede encontrar en la consola AWS IoT, en Configuración con greengrass. como prefijo.
keepAlive	El periodo de MQTT KeepAlive en segundos.	Se trata de un valor opcional. El valor predeterminado es 600 segundos.

Campo	Descripción	Notas
`useSystemd`	Una marca binaria si el dispositivo usa `systemd`.	Los valores son `yes` o `no`. Utilice el script de dependencias del Módulo 1 para comprobar si el dispositivo usa `systemd`.
`managedRespawn`	Una función de actualizaciones opcional over-the-air (OTA) que indica que el agente de OTA debe ejecutar un código personalizado antes de realizar una actualización.	Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core.

GGC v1.1

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   }
}
```

El archivo `config.json` existe en */greengrass-root*`/config` y contiene los parámetros siguientes:

Campo	Descripción	Notas
`caPath`	La ruta a la CA raíz de AWS IoT relativa a la carpeta */greengrass-root* / `certs`.	Guarde el archivo en la carpeta */greengrass-root*/`certs`.
`certPath`	La ruta del certificado de AWS IoT Greengrass del núcelo de la carpeta */greengrass-root* / `certs`.	Guarde el archivo en la carpeta */greengrass-root*/`certs`.
`keyPath`	La ruta de la clave privada de AWS IoT Greengrass del núcleo de la carpeta */greengrass-root* / `certs`.	Guarde el archivo en la carpeta */greengrass-root*/`certs`.
`thingArn`	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que representa el AWS IoT Greengrass del núcleo.	Este valor se puede encontrar en la consola AWS IoT Greengrass, en la definición de su objeto de AWS IoT.
`iotHost`	El punto de enlace de AWS IoT.	Este valor se puede encontrar en la consola AWS IoT, en Configuración.
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Este valor se puede encontrar en la consola AWS IoT, en Configuración con `greengrass.` como prefijo.
`keepAlive`	El periodo de MQTT `KeepAlive` en segundos.	Se trata de un valor opcional. El valor predeterminado es 600 segundos.

Campo	Descripción	Notas
`useSystemd`	Una marca binaria si el dispositivo usa `systemd`.	Los valores son `yes` o `no`. Utilice el script de dependencias del Módulo 1 para comprobar si el dispositivo usa `systemd`.

GGC v1.0

En AWS IoT Greengrass Core v1.0, `config.json` está implementado en *`greengrass-root`*`/configuration`.

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   }
}
```

El archivo `config.json` existe en */`greengrass-root`*`/configuration` y contiene los parámetros siguientes:

Campo	Descripción	Notas
`caPath`	La ruta a la CA raíz de AWS IoT relativa a la carpeta */`greengrass-root` /*	Guarde el archivo en la carpeta */`greengrass-root`*`/configuration/certs` .

Campo	Descripción	Notas
	`configuration/certs` .	
`certPath`	La ruta del certificado de AWS IoT Greengrass del núcelo de la carpeta */greengrass-root* / `configuration/certs` .	Guarde el archivo en la carpeta */greengrass-root*`/configuration/certs` .
`keyPath`	La ruta de la clave privada de AWS IoT Greengrass del núcleo de la carpeta */greengrass-root* / `configuration/certs` .	Guarde el archivo en la carpeta */greengrass-root*`/configuration/certs` .
`thingArn`	El Nombre de recurso de Amazon (ARN) del objeto de AWS IoT que representa el AWS IoT Greengrass del núcleo.	Este valor se puede encontrar en la consola AWS IoT Greengrass, en la definición de su objeto de AWS IoT.
`iotHost`	El punto de enlace de AWS IoT.	Este valor se puede encontrar en la consola AWS IoT, en Configuración.
`ggHost`	El punto de enlace de AWS IoT Greengrass.	Este valor se puede encontrar en la consola AWS IoT, en Configuración con `greengrass.` como prefijo.
`keepAlive`	El periodo de MQTT `KeepAlive` en segundos.	Se trata de un valor opcional. El valor predeterminado es 600 segundos.

Campo	Descripción	Notas
`useSystemd`	Una marca binaria si el dispositivo usa `systemd`.	Los valores son `yes` o `no`. Utilice el script de dependencias del Módulo 1 para comprobar si el dispositivo usa `systemd`.

Los puntos de conexión del servicio deben coincidir con el tipo de certificado de CA raíz.

Los puntos de enlace de AWS IoT Core y AWS IoT Greengrass deben corresponder al tipo de certificado de entidad de certificación raíz del dispositivo. Si los puntos de enlace y el tipo de certificado no coinciden, se produce un error en los intentos de autenticación entre el dispositivo y AWS IoT Core o AWS IoT Greengrass. Para obtener más información, consulte Autenticación del servidor en la Guía del desarrollador de AWS IoT.

Si su dispositivo utiliza un certificado CA raíz de Amazon Trust Services (ATS), que es el método preferido, también deberá utilizar puntos de conexión ATS para las operaciones del plano de datos de administración y descubrimiento de dispositivos. Los puntos de enlace de ATS incluyen el segmento `ats`, como se muestra en la siguiente sintaxis del punto de enlace de AWS IoT Core.

```
prefix-ats.iot.region.amazonaws.com
```

> **Note**
>
> Por motivos de compatibilidad con versiones anteriores, AWS IoT Greengrass actualmente es compatible con los certificados de CA VeriSign raíz y los puntos de conexión antiguos en algunos casosRegión de AWS. Si utiliza un certificado de CA VeriSign raíz heredado, le recomendamos que cree un punto de conexión ATS y, en su lugar, utilice un certificado de CA raíz de ATS. De lo contrario, asegúrese de utilizar los puntos de enlace heredados correspondientes. Para obtener más información, consulte Puntos de enlace heredados admitidos en la Referencia general de Amazon Web Services.

Puntos de enlace en config.json

En un dispositivo central de Greengrass, los puntos de enlace se especifican en el objeto `coreThing` del archivo `config.json`. La propiedad `iotHost` representa el punto de enlace de AWS IoT Core. La propiedad `ggHost` representa el punto de enlace de AWS IoT Greengrass. En el siguiente fragmento de código de ejemplo, estas propiedades especifican puntos de enlace de ATS.

```
{
  "coreThing" : {
    ...
    "iotHost" : "abcde1234uwxyz-ats.iot.us-west-2.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
    ...
  },
```

Punto de enlace de AWS IoT Core

Para obtener su punto de enlace de AWS IoT Core, ejecute el comando aws iot describe-endpoint de la CLI con el parámetro `--endpoint-type` adecuado.

- Para devolver un punto de enlace firmado por ATS, ejecute:

  ```
  aws iot describe-endpoint --endpoint-type iot:Data-ATS
  ```

- Para devolver un punto final VeriSign firmado heredado, ejecuta:

  ```
  aws iot describe-endpoint --endpoint-type iot:Data
  ```

Punto de enlace de AWS IoT Greengrass

Su punto de enlace de AWS IoT Greengrass es su punto de enlace `iotHost` con el prefijo de host reemplazado por greengrass. Por ejemplo, el punto de enlace firmado por ATS es `greengrass-ats.iot.`*`region`*`.amazonaws.com`. Utiliza la misma región que el punto de enlace de AWS IoT Core.

Realizar la conexión en el puerto 443 o a través de un proxy de red

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

Los núcleos de Greengrass se comunican con AWS IoT Core utilizando el protocolo de mensajería MQTT con autenticación de cliente TLS. Convencionalmente, MQTT sobre TLS utiliza el puerto

8883. Sin embargo, como medida de seguridad, los entornos restrictivos podrían limitar el tráfico de entrada y salida a un pequeño rango de puertos TCP. Por ejemplo, el firewall de una compañía podría abrir el puerto 443 para el tráfico HTTPS, pero cerrar otros puertos que se utilizan para protocolos menos frecuentes, como, por ejemplo, el puerto 8883 para tráfico MQTT. Otros entornos restrictivos podrían requerir que todo el tráfico pase a través de un proxy HTTP antes de conectarse a Internet.

Para habilitar la comunicación en estos casos, AWS IoT Greengrass permite las siguientes configuraciones:

- MQTT con autenticación del cliente de TLS a través del puerto 443. Si su red permite realizar conexiones al puerto 443, puede configurar el núcleo para usar el puerto 443 para tráfico MQTT en lugar del puerto predeterminado 8883. Esto puede ser una conexión directa al puerto 443 o una conexión a través de un servidor proxy de red.

  AWS IoT Greengrass utiliza la extensión TLS red de protocolo de capa de aplicación (ALPN) para habilitar esta conexión. Como en el caso de la configuración predeterminada, MQTT sobre TLS en el puerto 443 utiliza la autenticación de cliente basada en certificados.

  Cuando se configura para usar una conexión directa al puerto 443, el núcleo admite actualizaciones de AWS IoT Greengrass software over-the-air (OTA). Esta compatibilidad requiere AWS IoT Greengrass Core v1.9.3 o posterior.
- Comunicación HTTPS a través del puerto 443. AWS IoT Greengrass envía el tráfico HTTPS a través del puerto 8443 de forma predeterminada, pero puede configurarlo para que utilice el puerto 443.
- Conexión a través de un proxy de red. Puede configurar un servidor proxy de red para que actúe como intermediario para conectarse al núcleo de Greengrass. Solo se admiten la autenticación básica y los proxies HTTP y HTTPS.

  La configuración del proxy se transfiere a funciones de Lambda definidas por el usuario a través de las variables de entorno `http_proxy`, `https_proxy`, y `no_proxy`. Las funciones de Lambda definidas por el usuario deben utilizar estos ajustes de transferencia para conectarse a través del proxy. Las bibliotecas comunes utilizadas por las funciones de Lambda para establecer la conexión (como boto3 o cURL y los paquetes `requests` de python) suelen utilizar estas variables de entorno de forma predeterminada. Si una función de Lambda especifica también estas mismas variables de entorno, AWS IoT Greengrass no las invalida.

> **Important**
>
> Los núcleos de Greengrass que están configurados para usar un proxy de red no admiten actualizaciones OTA.

Para configurar MQTT a través del puerto 443

Esta característica requiere AWS IoT Greengrass Core versión 1.7 o posterior.

Este procedimiento permite que el núcleo de Greengrass utilice el puerto 443 para la mensajería MQTT con AWS IoT Core.

1. Ejecute el siguiente comando para detener el daemon de Greengrass.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

2. Abra *greengrass-root*`/config/config.json` para editarlo como usuario.
3. En el objeto `coreThing`, añada la propiedad `iotMqttPort` y establezca el valor en **443**, tal y como se muestra en el siguiente ejemplo.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "iotMqttPort" : 443,
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "keepAlive" : 600
    },
    ...
}
```

4. Inicie el daemon.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

Para configurar HTTPS a través del puerto 443

Esta característica requiere AWS IoT Greengrass Core versión 1.8 o posterior.

Este procedimiento configura el núcleo para usar el puerto 443 para la comunicación HTTPS.

1. Ejecute el siguiente comando para detener el daemon de Greengrass.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

2. Abra *greengrass-root*`/config/config.json` para editarlo como usuario.
3. En el objeto `coreThing`, añada las propiedades `iotHttpPort` e `ggHttpPort`, tal y como se muestra en el siguiente ejemplo.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "iotHttpPort" : 443,
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "ggHttpPort" : 443,
        "keepAlive" : 600
    },
    ...
}
```

4. Inicie el daemon.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

Para configurar un proxy de red

Esta característica requiere AWS IoT Greengrass Core versión 1.7 o posterior.

Este procedimiento permite a AWS IoT Greengrass conectarse a Internet a través de un proxy de red HTTP o HTTPS.

1. Ejecute el siguiente comando para detener el daemon de Greengrass.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

2. Abra *greengrass-root*/config/config.json para editarlo como usuario.

3. En el objeto coreThing, añada el objeto networkProxy, tal y como se muestra en el siguiente ejemplo.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "keepAlive" : 600,
        "networkProxy": {
            "noProxyAddresses" : "http://128.12.34.56,www.mywebsite.com",
            "proxy" : {
                "url" : "https://my-proxy-server:1100",
                "username" : "Mary_Major",
                "password" : "pass@word1357"
            }
        }
    },
    ...
}
```

4. Inicie el daemon.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

Objeto networkProxy

Utilice el objeto `networkProxy` para especificar información sobre el proxy de red. Este objeto incluye las siguientes propiedades.

Campo	Descripción
`noProxyAddresses`	Opcional. Una lista separada por comas de direcciones IP o nombres de host que están exentos del proxy.

`proxy`	El proxy al que conectar. Un proxy tiene las siguientes propiedades.
	• `url`. La dirección URL del servidor proxy, en el formato `scheme://userinfo@host:port` .
	  • `scheme`. El esquema. Debe ser `http` o `https`.
	  • `userinfo`. Opcional. La información de nombre de usuario y contraseña. Si se especifica, los campos `username` y `password` se omiten.
	  • `host`. Nombre de host o dirección IP del servidor proxy.
	  • `port`. Opcional. El número de puerto. Si no se especifica, se usarán los valores predeterminados siguientes:
	  • `http`: 80
	  • `https`: 443
	• `username`. Opcional. El nombre de usuario que utilizar para autenticarse al servidor proxy.
	• `password`. Opcional. La contraseña que utilizar para autenticarse al servidor proxy.

Permitir puntos de enlace

La comunicación entre los dispositivos Greengrass y AWS IoT Core o AWS IoT Greengrass debe autenticarse. Esta autenticación se basa en certificados de dispositivo X.509 registrados y claves criptográficas. Para permitir que las solicitudes autenticadas se transmitan a través de proxies sin cifrado adicional, permita los puntos de enlace siguientes.

punto de enlace	Puerto	Descripción
`greengrass.` *`region`*`.amazonaws.com`	443	Se utiliza para las operaciones del plano de control para la administración de grupos.
`prefix``-ats.iot.` *`region`*`.amazonaws.com` o *`prefix`*`.iot.`*`region`*`.amazonaws.com`	MQTT: 8883 o 443 HTTPS: 8443 o 443	Se utiliza para las operaciones del plano de datos para la administración de dispositivos, como la sincronización de sombras. Permita el uso uno o ambos puntos de conexión, en función de si el núcleo y

punto de enlace	Puerto	Descripción
		los dispositivos cliente utilizan certificados de entidad de certificación raíz de Amazon Trust Services (opción preferida), certificados de entidad de certificación raíz antiguos o ambos. Para obtener más información, consulte the section called "Los puntos de conexión del servicio deben coincidir con el tipo de certificado".

punto de enlace	Puerto	Descripción

`greengrass-ats.iot. region.amazonaws.com`
 	
o
 	
`greengrass.iot. region.amazonaws.com`
	8443 o 443	Se utiliza para las operaciones de detección de dispositivos.
		
		Permita el uso uno o ambos puntos de conexión, en función de si el núcleo y los dispositivos cliente utilizan certificados de entidad de certificación raíz de Amazon Trust Services (opción preferida), certificados de entidad de certificación raíz antiguos o ambos. Para obtener más información, consulte the

punto de enlace	Puerto	Descripción
		section called “Los puntos de conexión del servicio deben coincidir con el tipo de certificado”. Note Los clientes que se conecten por el puerto 443 deben implementar la extensión TLS de Negociación de Protocolo de Capa de Aplicación (ALPN) y

punto de enlace	Puerto	Descripción
		pasar `x-amzn-http-ca` como el `ProtocolName` en el `ProtocolNameList`. Para obtener más información, consulte Protocolos en la Guía para desarrolladores de AWS IoT.

punto de enlace	Puerto	Descripción
`*.s3.amazonaws.com`	443	Se utiliza para las operaciones de implementación y over-the-air las actualizaciones. Este formato incluye el carácter *, porque los prefijos de punto de enlace se controlan internamente y pueden cambiar en cualquier momento.
`logs.region.amazonaws.com`	443	Obligatorio si el grupo de Greengrass está configurado para escribir los registros en CloudWatch.

Configurar un directorio de escritura para AWS IoT Greengrass

Esta característica está disponible para AWS IoT Greengrass Core versión 1.6 y posteriores.

De forma predeterminada, el software de AWS IoT Greengrass Core se implementa en un solo directorio raíz donde AWS IoT Greengrass realiza todas las operaciones de lectura y escritura. Sin embargo, puede configurar AWS IoT Greengrass para que utilice un directorio independiente para todas las operaciones de escritura, incluida la creación de directorios y archivos. En este caso, AWS IoT Greengrass utiliza dos directorios de nivel superior:

- El directorio *greengrass-root*, que puede dejar como lectura-escritura o, si lo desea, solo lectura. Contiene el software de AWS IoT Greengrass Core y otros componentes fundamentales que deben permanecer inmutables durante el tiempo de ejecución, como los certificados y `config.json`.
- El directorio de escritura especificado. Contiene contenido con permiso de escritura, como registros, información de estado y funciones de Lambda implementadas, definidas por el usuario.

Esta configuración da como resultado la siguiente estructura de directorios.

Directorio raíz de Greengrass

```
greengrass-root/
|-- certs/
|   |-- root.ca.pem
|   |-- hash.cert.pem
|   |-- hash.private.key
|   |-- hash.public.key
|-- config/
|   |-- config.json
|-- ggc/
|   |-- packages/
|       |-- package-version/
|           |-- bin/
|               |-- daemon
|           |-- greengrassd
|           |-- lambda/
|           |-- LICENSE/
|           |-- release_notes_package-version.html
|               |-- runtime/
|                   |-- java8/
|                   |-- nodejs8.10/
```

```
|                   |-- python3.8/
|   |-- core/
```

Directorio de escritura

```
write-directory/
|-- packages/
|   |-- package-version/
|       |-- ggc_root/
|       |-- rootfs_nosys/
|       |-- rootfs_sys/
|       |-- var/
|-- deployment/
|   |-- group/
|       |-- group.json
|   |-- lambda/
|   |-- mlmodel/
|-- var/
|   |-- log/
|   |-- state/
```

Para configurar un directorio de escritura

1. Ejecute el siguiente comando para detener el daemon de AWS IoT Greengrass:

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd stop
   ```

2. Abra *greengrass-root*`/config/config.json` para editarlo como usuario.
3. Añada `writeDirectory` como parámetro y especifique la ruta al directorio de destino, tal y como se muestra en el siguiente ejemplo.

   ```
   {
       "coreThing": {
           "caPath": "root-CA.pem",
           "certPath": "hash.pem.crt",
           ...
       },
       ...
   ```

```
    "writeDirectory" : "/write-directory"
}
```

> **Note**
>
> Puede actualizar la configuración `writeDirectory` tantas veces como desee. Después de actualizar la configuración, AWS IoT Greengrass utiliza el directorio de escritura recién especificado en el siguiente inicio, pero no migra contenido desde el directorio de escritura anterior.

4. Ahora que el directorio de escritura está configurado, tiene la opción de hacer que el directorio *greengrass-root* sea de solo lectura. Para obtener instrucciones, consulte Cómo hacer que el directorio raíz de Greengrass sea de solo lectura.

   De lo contrario, inicie el daemon de AWS IoT Greengrass:

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd start
   ```

Para hacer que el directorio raíz de Greengrass sea de solo lectura

Siga estos pasos solo si desea hacer el directorio raíz de Greengrass de solo lectura. El directorio de escritura debe estar configurado antes de comenzar.

1. Conceda permisos de acceso a directorios necesarios:

   a. Otorgue permisos de lectura y escritura al propietario de `config.json`.

      ```
      sudo chmod 0600 /greengrass-root/config/config.json
      ```

   b. Haga a ggc_user el propietario de los certificados y los directorios Lambda del sistema.

      ```
      sudo chown -R ggc_user:ggc_group /greengrass-root/certs/
      sudo chown -R ggc_user:ggc_group /greengrass-root/ggc/packages/1.11.6/lambda/
      ```

> **Note**
>
> Las cuentas ggc_user y ggc_group se utilizan de forma predeterminada para ejecutar funciones de Lambda del sistema. Si ha configurado la identidad de acceso predeterminada de nivel de grupo para utilizar distintas cuentas, debe proporcionar permisos a dicho usuario (UID) y grupo (GID) en su lugar.

2. Haga que el directorio raíz *greengrass-root* sea de solo lectura empleando el mecanismo que prefiera.

> **Note**
>
> Una forma para que el directorio raíz *greengrass-root* sea de solo lectura es montar el directorio como de solo lectura. Sin embargo, para aplicar las actualizaciones over-the-air (OTA) al software AWS IoT Greengrass principal de un directorio montado, primero se debe desmontar el directorio y volver a montarlo después de la actualización. Puede añadir estas operaciones `umount` y `mount` a los scripts `ota_pre_update` y `ota_post_update`. Para obtener más información sobre las actualizaciones OTA, consulte the section called “Agente de actualización de OTA para Greengrass” y the section called “Regeneración administrada con actualizaciones OTA”.

3. Inicie el daemon.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

Si los permisos del paso 1 no están establecidos correctamente, el daemon no se iniciará.

Configuración de MQTT

En el entorno de AWS IoT Greengrass, los dispositivos cliente locales, las funciones de Lambda, los conectores y los componentes del sistema pueden comunicarse entre sí y con AWS IoT Core. Toda la comunicación pasa por el núcleo, que gestiona las suscripciones que autorizan la comunicación MQTT entre entidades.

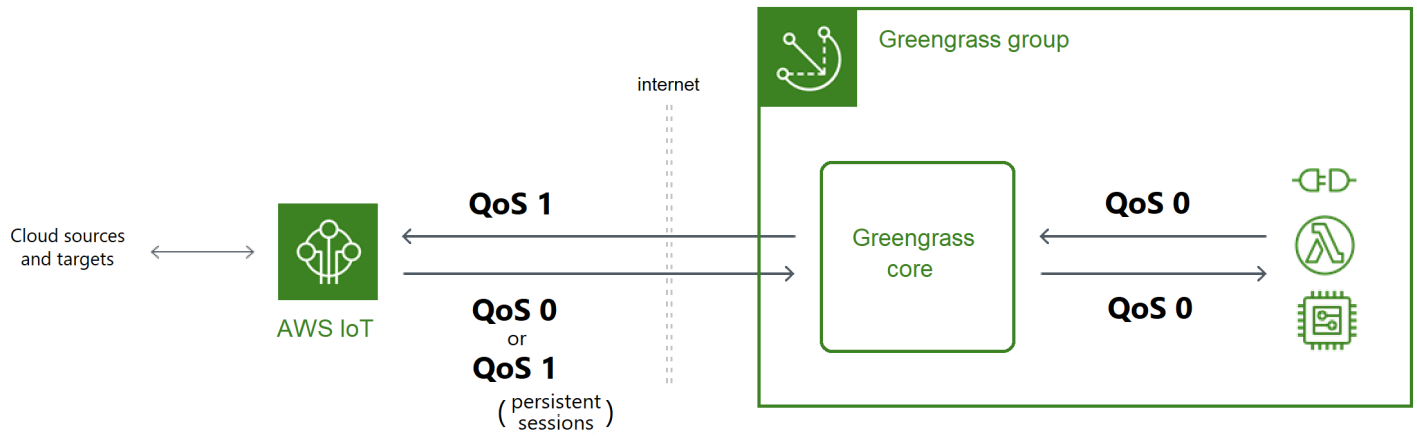
Para obtener información sobre la configuración de MQTT para la que puede configurar AWS IoT Greengrass, consulte las siguientes secciones:

- the section called “Mensaje de calidad del servicio”
- the section called “Cola de mensajes MQTT”
- the section called “Sesiones persistentes de MQTT con AWS IoT Core”
- the section called “ID de cliente para conexiones MQTT con AWS IoT”
- Puerto MQTT para la mensajería local
- the section called “Tiempo de espera para las operaciones de publicación, suscripción y cancelación de suscripción en las conexiones MQTT con la Nube de AWS”

> **Note**
>
> OPC-UA es un estándar de intercambio de información para la comunicación industrial. Para implementar el soporte para OPC-UA en el núcleo de Greengrass, puede usar el conector IoT. SiteWise El conector envía datos de dispositivos industriales desde servidores de OPC-UA a propiedades de recursos en AWS IoT SiteWise.

Mensaje de calidad del servicio

AWS IoT Greengrass admite niveles de calidad de servicio (QoS) 0 o 1, dependiendo de su configuración y del destino y la dirección de la comunicación. El núcleo de Greengrass actúa como cliente para la comunicación con AWS IoT Core y como agente de mensajes para la comunicación en la red local.



Para obtener más información sobre MQTT y QoS, consulte Introducción en el sitio web de MQTT.

Comunicación con el Nube de AWS

- Los mensajes salientes usan QoS 1

  El núcleo envía mensajes destinados a los destinos de Nube de AWS mediante QoS 1. AWS IoT Greengrass utiliza una cola de mensajes MQTT para procesar estos mensajes. Si AWS IoT no confirma la entrega de mensajes, el mensaje se pone en cola para intentarlo más tarde. No se puede volver a intentar enviar el mensaje si la cola está llena. La confirmación de entrega del mensaje puede ayudar a minimizar la pérdida de datos provocada por la conectividad intermitente.

  Como los mensajes salientes a AWS IoT utilizan QoS 1, la velocidad máxima a la que el núcleo de Greengrass puede enviar mensajes depende de la latencia entre el núcleo y AWS IoT. Cada vez que el núcleo envía un mensaje, espera a que AWS IoT confirme el mensaje antes de enviar el siguiente mensaje. Por ejemplo, si el tiempo de ida y vuelta entre el núcleo y su Región de AWS es de 50 milisegundos, el núcleo puede enviar hasta 20 mensajes por segundo. Tenga en cuenta este comportamiento al elegir la Región de AWS donde se conecta el núcleo. Para transferir datos de IoT de gran volumen a la Nube de AWS, puede usar el administrador de secuencias.

  Para obtener más información sobre la cola de mensajes MQTT, incluyendo cómo configurar una caché de almacenamiento local que pueda persistir los mensajes destinados a los objetivos Nube de AWS, consulte the section called “Cola de mensajes MQTT”.

- Los mensajes entrantes usan QoS 0 (predeterminado) o QoS 1

  De forma predeterminada, el núcleo se suscribe con QoS 0 a mensajes de fuentes de la Nube de AWS. Si habilita sesiones persistentes, el núcleo se suscribe con QoS 1. Esto puede ayudar a minimizar la pérdida de datos debido a la conectividad intermitente. Para administrar la QoS para estas suscripciones, configure la configuración de persistencia en el componente del sistema de cola de impresión local.

  Para obtener más información, incluida la forma de habilitar el núcleo para establecer una sesión persistente con los destinos de la Nube de AWS, consulte the section called “Sesiones persistentes de MQTT con AWS IoT Core”.

Comunicación con objetivos locales

Todas las comunicaciones locales usan QoS 0. El núcleo intenta enviar un mensaje a un destino local, que puede ser una función de Lambda de Greengrass, un conector o un dispositivo cliente.

El núcleo no almacena mensajes ni confirma entrega. Los mensajes se pueden dejar en cualquier lugar entre los componentes.

> **Note**
>
> Aunque la comunicación directa entre funciones de Lambda no utiliza mensajería MQTT, el comportamiento es el mismo.

Cola de mensajes MQTT para objetivos en la nube

Los mensajes MQTT que se destinan a destinos en la Nube de AWS se ponen en la cola a la espera de procesamiento. Los mensajes en la cola se procesan siguiendo el orden primero en entrar, primero en salir (FIFO). Después de procesar un mensaje y publicarlo en AWS IoT Core, el mensaje se elimina de la cola.

Por defecto, el núcleo Greengrass almacena en memoria los mensajes no procesados destinados a los objetivos en la Nube de AWS. En su lugar, puede configurar el núcleo para almacenar mensajes sin procesar en una caché de almacenamiento local. A diferencia del almacenamiento en memoria, el almacenamiento en caché local tiene la capacidad de persistir a través de reinicios del núcleo (por ejemplo, después de la implementación de un grupo o reinicio de un dispositivo), por lo que AWS IoT Greengrass puede continuar procesando mensajes. También puede configurar el tamaño del almacenamiento.

> **Warning**
>
> El núcleo de Greengrass puede poner en cola los mensajes MQTT duplicados cuando pierde la conexión, ya que vuelve a intentar una operación de publicación antes de que el cliente MQTT detecte que está desconectado. Para evitar la duplicación de mensajes MQTT para los destinos en la nube, configure el valor del núcleo en menos de la `keepAlive` mitad de su valor `mqttOperationTimeout`. Para obtener más información, consulte Archivo de configuración de AWS IoT Greengrass Core.

AWS IoT Greengrass utiliza el componente del sistema de cola de impresión (la función de Lambda `GGCloudSpooler`) para administrar la cola de mensajes. Puede utilizar las siguientes variables de entorno `GGCloudSpooler` para configurar los valores de almacenamiento.

- GG_CONFIG_STORAGE_TYPE. La ubicación de la cola de mensajes. Los siguientes valores son válidos:
  - `FileSystem`. Almacene mensajes sin procesar en la caché de almacenamiento local en el disco del dispositivo central físico. Cuando el núcleo se reinicia, se conservan los mensajes que hay en cola a la espera de procesamiento. Los mensajes se eliminan después de que se procesan.
  - `Memory` (predeterminada). Almacena los mensajes sin procesar en la memoria. Cuando el núcleo se reinicia, los mensajes en la cola se pierden.

    Esta opción está optimizada para dispositivos con capacidades limitadas de hardware. Si se utiliza esta configuración, le recomendamos que implemente grupos o reinicie el dispositivo en un momento en el que la interrupción del servicio sea mínima.

- GG_CONFIG_MAX_SIZE_BYTES. El tamaño de almacenamiento en bytes. Este valor puede ser cualquier número entero que no sea negativo superior o igual a 262144 (256 KB); un tamaño más pequeño impide que se inicie el software de AWS IoT Greengrass Core. El valor predeterminado es 2.5 MB de tamaño. Cuando se alcanza el límite de tamaño, los mensajes en cola más antiguos se sustituyen por mensajes nuevos.

> **Note**
>
> Esta característica está disponible para AWS IoT Greengrass Core versión v1.6 y posteriores. Las versiones anteriores utilizan almacenamiento en memoria con un tamaño de cola de 2,5 MB. No puede ajustar la configuración de almacenamiento para versiones anteriores.

Para almacenar los mensajes en la caché local

Puede configurar AWS IoT Greengrass para almacenar los mensajes en caché en el sistema de archivos de modo que persistan entre los reinicios del núcleo. Para ello, se implementa una versión de definición de característica en la que la característica `GGCloudSpooler` establece el tipo de almacenamiento en `FileSystem`. Debe utilizar la API de AWS IoT Greengrass para configurar el almacenamiento en caché local. No puede hacer esto en la consola.

En el procedimiento siguiente, se usa el comando de CLI `create-function-definition-version` para configurar la cola de impresión para que guarde los mensajes que hay en cola en el sistema de archivos. También configura un tamaño de cola de 2.6 MB.

1. Obtenga los ID de la versión de grupo y grupo de Greengrass de destino. En este procedimiento, suponemos que estos son el último grupo y la última versión de grupo. La siguiente consulta devuelve el grupo creado más recientemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   También puede hacer la consulta por nombre. No es necesario que los nombres de grupo sean únicos, por lo que podrían devolverse varios grupos.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

2. Copie los valores `Id` y `LatestVersion` del grupo de destino en la salida.
3. Obtenga la última versión del grupo.

   - Reemplace *`id-grupo`* con el `Id` que ha copiado.
   - Reemplace *`latest-group-version-id`* por el `LatestVersion` que ha copiado.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id latest-group-version-id
   ```

4. Desde el objeto `Definition` de la salida, copie el `CoreDefinitionVersionArn` y los ARN de todos los demás componentes de grupo excepto `FunctionDefinitionVersionArn`. Utilizará estos valores cuando cree una nueva versión del grupo.

5. Desde `FunctionDefinitionVersionArn` en el resultado, copie el ID de la definición de la característica. El ID es el GUID que va detrás del segmento `functions` en el ARN, tal y como se muestra en el siguiente ejemplo.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> O puede crear una definición de la función ejecutando el comando `create-function-definition` y luego copiar el ID del resultado.

6. Añada una versión de definición de la función a la definición de la característica.

   - *`function-definition-id`*Sustitúyalo por el `Id` que copiaste para la definición de la función.
   - *`arbitrary-function-id`*Sustitúyalo por un nombre para la función, por ejemplo**`spooler-function`**.
   - Agregue las funciones de Lambda que desea incluir en esta versión en la matriz `functions`. Puede utilizar el comando `get-function-definition-version` para obtener las funciones de Lambda de Greengrass de una versión de la definición de característica existente.

> **Warning**
>
> Asegúrese de que se especifica un valor para `GG_CONFIG_MAX_SIZE_BYTES` que sea mayor o igual a 262144. Un tamaño más pequeño impide que se inicie el software de AWS IoT Greengrass Core.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[{"FunctionArn":
 "arn:aws:lambda:::function:GGCloudSpooler:1","FunctionConfiguration":
 {"Environment": {"Variables":
{"GG_CONFIG_MAX_SIZE_BYTES":"2621440","GG_CONFIG_STORAGE_TYPE":"FileSystem"}},"Executable":
```

```
 "spooler","MemorySize": 32768,"Pinned": true,"Timeout": 3},"Id": "arbitrary-
function-id"}]'
```

> **Note**
>
> Si previamente configuró la variable de entorno `GG_CONFIG_SUBSCRIPTION_QUALITY` para admitir sesiones persistentes con AWS IoT Core, inclúyala en esta instancia de función.

7. Copie el `Arn` de la versión de definición de la característica del resultado.
8. Cree una versión de grupo que contenga la función de Lambda del sistema.

   - Reemplace *`id-grupo`* con el `Id` del grupo.
   - *`core-definition-version-arn`*Sustitúyalo por el `CoreDefinitionVersionArn` que copió de la última versión del grupo.
   - *`function-definition-version-arn`*Sustitúyala por la `Arn` que copió para la nueva versión de definición de funciones.
   - Reemplace los ARN para otros componentes del grupo (por ejemplo, `SubscriptionDefinitionVersionArn` o `DeviceDefinitionVersionArn`) que ha copiado de la última versión del grupo.
   - Elimine los parámetros no utilizados. Por ejemplo, elimine `--resource-definition-version-arn` si la versión de su grupo no contiene ningún recurso.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Copie la `Version` del resultado. Este es el ID de la nueva versión del grupo.
10. Implemente el grupo con la nueva versión del grupo.

    - Reemplace *`id-grupo`* con el `Id` que ha copiado para el grupo.
    - *`group-version-id`*Sustitúyalo por el `Version` que copió para la nueva versión de grupo.

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

Para actualizar la configuración de almacenamiento, utilice la API de AWS IoT Greengrass para crear una nueva versión de la definición de la característica que contenga la característica `GGCloudSpooler` con la configuración actualizada. A continuación, añada la versión de la definición de la característica a una nueva versión del grupo (junto con los demás componentes del grupo) e implemente la versión del grupo. Si desea restaurar la configuración predeterminada, puede implementar una versión de definición de la característica que no incluya la característica `GGCloudSpooler`.

Esta función de Lambda del sistema no es visible en la consola de . Sin embargo, después de que la función se añade a la última versión del grupo, se incluye en las implementaciones que se realizan desde la consola (a menos que utilice la API para sustituirla o eliminarla).

Sesiones persistentes de MQTT con AWS IoT Core

Esta característica está disponible para AWS IoT Greengrass Core v1.10 y versiones posteriores.

Un núcleo de Greengrass puede establecer una sesión persistente con el agente de mensajes de AWS IoT. Una sesión persistente es una conexión continua que permite al núcleo recibir mensajes enviados mientras el núcleo está fuera de línea. El núcleo es el cliente en la conexión.

En una sesión persistente, el agente de mensajes AWS IoT guarda todas las suscripciones que realiza el núcleo durante la conexión. Si el núcleo se desconecta, el agente de mensajes AWS IoT almacena mensajes nuevos y no reconocidos publicados como QoS 1 y destinados a destinos locales, como funciones de Lambda y dispositivos cliente. Cuando el núcleo se vuelve a conectar, la sesión persistente se reanuda y el agente de mensajes de AWS IoT envía mensajes almacenados al núcleo a una velocidad máxima de 10 mensajes por segundo. Las sesiones persistentes tienen un período de caducidad predeterminado de 1 hora, que comienza cuando el agente de mensajes detecta que el núcleo se desconecta. Para obtener más información, consulte Sesiones persistentes de MQTT en la Guía del desarrollador de AWS IoT.

AWS IoT Greengrass utiliza el componente del sistema de cola de impresión (la función de Lambda `GGCloudSpooler`) para crear suscripciones que tengan AWS IoT como origen. Puede utilizar la siguiente variable de entorno `GGCloudSpooler` para configurar sesiones persistentes.

- GG_CONFIG_SUBSCRIPTION_QUALITY. La calidad de las suscripciones que tienen AWS IoT como origen. Los siguientes valores son válidos:
  - `AtMostOnce` (predeterminada). Deshabilita las sesiones persistentes. Las suscripciones usan QoS 0.
  - `AtLeastOncePersistent`. Habilita sesiones persistentes. Establece el indicador `cleanSession` a `0` en mensajes `CONNECT` y se suscribe con QoS 1.

    Se garantiza que los mensajes publicados con QoS 1 que recibe el núcleo lleguen a la cola de trabajo en memoria del daemon Greengrass. El núcleo reconoce el mensaje después de añadirlo a la cola. La comunicación posterior de la cola al destino local (por ejemplo, la función de Lambda de Greengrass, conector o dispositivo) se envía como QoS 0. AWS IoT Greengrass no garantiza la entrega a los destinos locales.

    > **Note**
    >
    > Puede utilizar la propiedad de configuración maxWorkItemCount para controlar el tamaño de la cola de elementos de trabajo. Por ejemplo, puede aumentar el tamaño de la cola si su carga de trabajo requiere mucho tráfico MQTT.

    Cuando las sesiones persistentes están habilitadas, el núcleo abre al menos una conexión adicional para el intercambio de mensajes MQTT con AWS IoT. Para obtener más información, consulte the section called “ID de cliente para conexiones MQTT con AWS IoT”.

Para configurar sesiones persistentes de MQTT

Puede configurar AWS IoT Greengrass para usar sesiones persistentes con AWS IoT Core. Para ello, se implementa una versión de definición de función en la que la función `GGCloudSpooler` establece la calidad de suscripción en `AtLeastOncePersistent`. Esta configuración se aplica a todas las suscripciones que tengan AWS IoT Core (`cloud`) como origen. Debe usar la API AWS IoT Greengrass para configurar sesiones persistentes. No puede hacer esto en la consola.

El siguiente procedimiento utiliza el comando de la CLI `create-function-definition-version` para configurar el spooler para que utilice sesiones persistentes. En este procedimiento suponemos que va a actualizar la configuración de la versión más reciente de un grupo existente.

1. Obtenga los ID de la versión de grupo y grupo de Greengrass de destino. En este procedimiento, suponemos que estos son el último grupo y la última versión de grupo. La siguiente consulta devuelve el grupo creado más recientemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   También puede hacer la consulta por nombre. No es necesario que los nombres de grupo sean únicos, por lo que podrían devolverse varios grupos.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

2. Copie los valores `Id` y `LatestVersion` del grupo de destino en la salida.
3. Obtenga la última versión del grupo.

   - Reemplace *`id-grupo`* con el `Id` que ha copiado.
   - Reemplace *`latest-group-version-id`* por el `LatestVersion` que ha copiado.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id latest-group-version-id
   ```

4. Desde el objeto `Definition` de la salida, copie el `CoreDefinitionVersionArn` y los ARN de todos los demás componentes de grupo excepto `FunctionDefinitionVersionArn`. Utilizará estos valores cuando cree una nueva versión del grupo.

5. Desde `FunctionDefinitionVersionArn` en el resultado, copie el ID de la definición de la característica. El ID es el GUID que va detrás del segmento `functions` en el ARN, tal y como se muestra en el siguiente ejemplo.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> O puede crear una definición de la función ejecutando el comando `create-function-definition` y luego copiar el ID del resultado.

6. Añada una versión de definición de la función a la definición de la característica.

   - *`function-definition-id`*Sustitúyala por la `Id` que copió para la definición de la función.
   - *`arbitrary-function-id`*Sustitúyalo por un nombre para la función, por ejemplo**`spooler-function`**.
   - Agregue las funciones de Lambda que desea incluir en esta versión en la matriz `functions`. Puede utilizar el comando `get-function-definition-version` para obtener las funciones de Lambda de Greengrass de una versión de la definición de característica existente.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[{"FunctionArn":
 "arn:aws:lambda:::function:GGCloudSpooler:1","FunctionConfiguration":
 {"Environment": {"Variables":
{"GG_CONFIG_SUBSCRIPTION_QUALITY":"AtLeastOncePersistent"}},"Executable":
 "spooler","MemorySize": 32768,"Pinned": true,"Timeout": 3},"Id": "arbitrary-
function-id"}]'
```

> **Note**
>
> Si previamente ha establecido las variables de entorno `GG_CONFIG_STORAGE_TYPE` o `GG_CONFIG_MAX_SIZE_BYTES` para definir la configuración de almacenamiento, inclúyalas en esta instancia de característica.

7. Copie el `Arn` de la versión de definición de la característica del resultado.
8. Cree una versión de grupo que contenga la función de Lambda del sistema.

   - Reemplace *`id-grupo`* con el `Id` del grupo.
   - *`core-definition-version-arn`*Sustitúyalo por el `CoreDefinitionVersionArn` que copió de la última versión del grupo.
   - *`function-definition-version-arn`*Sustitúyala por la `Arn` que copió para la nueva versión de definición de funciones.
   - Reemplace los ARN para otros componentes del grupo (por ejemplo, `SubscriptionDefinitionVersionArn` o `DeviceDefinitionVersionArn`) que ha copiado de la última versión del grupo.
   - Elimine los parámetros no utilizados. Por ejemplo, elimine `--resource-definition-version-arn` si la versión de su grupo no contiene ningún recurso.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Copie la `Version` del resultado. Este es el ID de la nueva versión del grupo.
10. Implemente el grupo con la nueva versión del grupo.

    - Reemplace *`id-grupo`* con el `Id` que ha copiado para el grupo.
    - *`group-version-id`*Sustitúyalo por el `Version` que copió para la nueva versión de grupo.

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

11. (Opcional) Aumente la propiedad maxWorkItemCount en el archivo de configuración principal. Esto puede ayudar al núcleo a manejar el mayor tráfico MQTT y la comunicación con los destinos locales.

Para actualizar la configuración de almacenamiento, utilice la API de AWS IoT Greengrass para crear una nueva versión de la definición de la característica que contenga la característica `GGCloudSpooler` con la configuración actualizada. A continuación, añada la versión de la definición de la característica a una nueva versión del grupo (junto con los demás componentes del grupo) e implemente la versión del grupo. Si desea restaurar la configuración predeterminada, puede crear una versión de definición de la función que no incluya la función `GGCloudSpooler`.

Esta función de Lambda del sistema no es visible en la consola de . Sin embargo, después de que la función se añade a la última versión del grupo, se incluye en las implementaciones que se realizan desde la consola (a menos que utilice la API para sustituirla o eliminarla).

ID de cliente para conexiones MQTT con AWS IoT

Esta característica está disponible para AWS IoT Greengrass Core versión 1.8 y posteriores.

El núcleo de Greengrass abre conexiones MQTT con AWS IoT Core para operaciones como la sincronización de sombras y la administración de certificados. Para estas conexiones, el núcleo genera los ID de cliente predecibles en función del nombre del objeto de núcleo. Los ID de cliente predecibles se puede utilizar con las características de monitorización, auditoría y precios, incluidos AWS IoT Device Defender y eventos de ciclo de vida de AWS IoT. También puede crear lógica en torno a los ID de cliente predecibles (por ejemplo, plantillas de política de suscripción basadas en atributos del certificado).

GGC v1.9 and later

Dos componentes del sistema Greengrass abren conexiones MQTT con AWS IoT Core. Estos componentes utilizan los siguientes patrones para generar los ID de cliente de las conexiones.

Operación	Patrón de ID de cliente
Implementaciones	*core-thing-name* Ejemplo: `MyCoreThing` Utilice este ID de cliente para conectar, desconectar, suscribir y cancelar la suscripción a notificaciones de eventos de ciclo de vida.
Suscripciones	*core-thing-name* `-c`*n*

Operación	Patrón de ID de cliente
	Ejemplo: `MyCoreThing-c01` *n* es un número entero que comienza en 00 y se incrementa con cada nueva conexión hasta un máximo de 250. El número de conexiones viene determinado por el número de dispositivos con los que sincroniza su estado sombra con AWS IoT Core (máximo 2.500 por grupo) y el número de suscripci ones con `cloud` que tienen como origen en el grupo (máximo 10.000 por grupo). El componente del sistema del administr ador de trabajos conecta con AWS IoT Core para intercambiar mensajes de las suscripci ones con un origen o destino en la nube. El administrador de trabajos también actúa como proxy para intercambiar mensajes entre AWS IoT Core, el servicio de sombra local y el administrador de certificados de dispositi vos.

Para calcular el número de conexiones MQTT por grupo, utilice la siguiente fórmula:

```
number of MQTT connections per group = number of connections for
Deployment Agent + number of connections for Subscriptions
```

Donde,

- número de conexiones para el agente de implementación = 1.
- número de conexiones para suscripciones =`(2 subscriptions for supporting certificate generation + number of MQTT topics in AWS IoT Core + number of device shadows synced) / 50`.
  - Donde, `50` = el número máximo de suscripciones por conexión que AWS IoT Core puede admitir.

> **Note**
>
> Si habilita las sesiones persistentes para la suscripción con AWS IoT Core, el núcleo abrirá al menos una conexión adicional para utilizarla en una sesión persistente. Los componentes del sistema no admiten sesiones persistentes, así que no pueden compartir esa conexión.

Para reducir el número de conexiones MQTT y ayudar a reducir los costes, puede utilizar las funciones de Lambda locales para agregar datos en la periferia. A continuación, envía los datos agregados a Nube de AWS. Como resultado, utiliza menos temas de MQTT en AWS IoT Core. Para más información, consulte Precios de AWS IoT Greengrass.

GGC v1.8

Varios componentes del sistema Greengrass abren conexiones MQTT con AWS IoT Core. Estos componentes utilizan los siguientes patrones para generar los ID de cliente de las conexiones.

Operación	Patrón de ID de cliente
Implementaciones	*core-thing-name* Ejemplo: `MyCoreThing` Utilice este ID de cliente para conectar, desconectar, suscribir y cancelar la suscripci ón a notificaciones de eventos de ciclo de vida.
Intercambio de mensajes MQTT con AWS IoT Core	*core-thing-name* `-spr` Ejemplo: `MyCoreThing-spr`
Sincronización de sombras	*core-thing-name* `-s`*nn* Ejemplo: `MyCoreThing-s01` *nn* es un número entero que comienza en 00 y aumenta con cada nueva conexión hasta un máximo de 03. El número de conexione

Operación	Patrón de ID de cliente
	s lo determina el número de dispositivos (un máximo de 200 dispositivos por grupo) que sincronizan su estado de sombra con AWS IoT Core (un máximo de 50 suscripciones por conexión).
Administración de certificados de dispositivo	*core-thing-name* `-dcm` Ejemplo: `MyCoreThing-dcm`

> **Note**
>
> La duplicación de ID de cliente en conexiones simultáneas puede provocar un bucle infinito de conexión-desconexión. Esto puede ocurrir si otro dispositivo está codificado para utilizar el nombre de dispositivo de núcleo como ID de cliente en las conexiones. Para obtener más información, consulte este paso de solución de problemas.

Los dispositivos de Greengrass también están plenamente integrados con el servicio de indexación de flotas de AWS IoT Device Management. Esto le permite indexar y buscar dispositivos basados en atributos de dispositivo, estado de sombra y estado de conexión en la nube. Por ejemplo, los dispositivos de Greengrass establecen al menos una conexión que utiliza el nombre de objeto como ID de cliente, a fin de que pueda utilizar la indexación de conectividad para detectar los dispositivos de Greengrass que están actualmente conectados a AWS IoT Core o desconectados de este. Para obtener más información, consulte Servicio de indexación de flotas en la Guía del desarrollador de AWS IoT.

Configuración del puerto MQTT para mensajería local

Esta característica requiere Core AWS IoT Greengrass versión 1.10 o posterior.

El núcleo de Greengrass actúa como agente de mensajes local para la mensajería MQTT entre funciones de Lambda locales, conectores y dispositivos cliente. De forma predeterminada, el núcleo utiliza el puerto 8883 para el tráfico MQTT en la red local. Es posible que desee cambiar el puerto para evitar un conflicto con otro software que se ejecute en el puerto 8883.

Para configurar el número de puerto que utiliza el núcleo para el tráfico MQTT local

1. Ejecute el siguiente comando para detener el daemon de Greengrass.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

2. Abra *greengrass-root*/config/config.json para editarlo como usuario.
3. En el objeto de `coreThing`, añada la propiedad de `ggMqttPort` y fije el valor en el número de puerto que desee utilizar. Los valores válidos son de 1024 a 65535. En el siguiente ejemplo se fija el número de puerto en `9000`.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "ggMqttPort" : 9000,
        "keepAlive" : 600
    },
    ...
}
```

4. Inicie el daemon.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

5. Si la detección automática de IP está habilitada para el núcleo, la configuración está completa.

   Si la detección automática de IP no está habilitada, debe actualizar la información de conectividad del núcleo. Esto permite que los dispositivos cliente reciban el número de puerto correcto durante las operaciones de detección para adquirir información de conectividad básica. Puede utilizar la consola AWS IoT o la API AWS IoT Greengrass para actualizar la información de conectividad básica. En este procedimiento, solo se actualiza el número de puerto. La dirección IP local del núcleo sigue siendo la misma.

Para actualizar la información de conectividad del núcleo (consola)

1. En la página de configuración del grupo, elija el núcleo de Greengrass.
2. En la página de detalles principales, elija la pestaña Puntos de conexión de agente MQTT.
3. Seleccione Gestionar puntos de conexión y, a continuación, seleccione Añadir punto de conexión
4. Introduzca su dirección IP local actual y el nuevo número de puerto. En el siguiente ejemplo se fija el número de puerto de `9000` para la dirección IP `192.168.1.8`.
5. Elimine el punto de enlace obsoleto y, a continuación, seleccione Update (Actualizar)

Para actualizar la información de conectividad del núcleo (API)

- Utilice la acción UpdateConnectivityInfo. En el siguiente ejemplo se utiliza `update-connectivity-info` en AWS CLI para fijar el número de puerto de `9000` para la dirección IP `192.168.1.8`.

```
aws greengrass update-connectivity-info \
    --thing-name "MyGroup_Core" \
    --connectivity-info "[{\"Metadata\":\"\",\"PortNumber\":9000,
\"HostAddress\":\"192.168.1.8\",\"Id\":\"localIP_192.168.1.8\"},{\"Metadata
\":\"\",\"PortNumber\":8883,\"HostAddress\":\"127.0.0.1\",\"Id\":
\"localhost_127.0.0.1_0\"}]"
```

> **Note**
>
> También puede configurar el puerto que utiliza el núcleo para la mensajería MQTT con AWS IoT Core. Para obtener más información, consulte the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.

Tiempo de espera para las operaciones de publicación, suscripción y cancelación de suscripción en las conexiones MQTT con la Nube de AWS

Esta característica está disponible en AWS IoT Greengrass v1.10.2 o posterior.

Puede configurar la cantidad de tiempo (en segundos) para permitir que el núcleo de Greengrass complete una operación de publicación, suscripción o cancelación de suscripción en las conexiones MQTT con AWS IoT Core. Puede que desee ajustar esta configuración si se agota el tiempo de espera de las operaciones debido a restricciones de ancho de banda o a una alta latencia. Para establecer esta configuración en el archivo config.json, agregue o cambie la propiedad `mqttOperationTimeout` en el objeto `coreThing`. Por ejemplo:

```
{
  "coreThing": {
    "mqttOperationTimeout": 10,
    "caPath": "root-ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    ...
  },
  ...
}
```

El tiempo de espera predeterminado es de 5 segundos. El tiempo de espera mínimo es de 5 segundos.

Activación de la detección automática de IP

Puede configurar AWS IoT Greengrass para permitir que los dispositivos cliente de un grupo de Greengrass descubran automáticamente el núcleo de Greengrass. Cuando está activado, el núcleo vigila los cambios en sus direcciones IP. Si una dirección cambia, el núcleo publica una lista actualizada de direcciones. Estas direcciones se ponen a disposición de los dispositivos cliente que estén en el mismo grupo de Greengrass que el núcleo.

> **Note**
>
> La política AWS IoT para los dispositivos cliente debe conceder el permiso `greengrass:Discover` que permita a los dispositivos extraer la información de conectividad del núcleo. Para obtener más información sobre esta instrucción de política, consulte the section called “Autorización de detección”.

Para activar esta característica desde la consola AWS IoT Greengrass, elija Detección automática cuando implementación su grupo de Greengrass por primera vez. También puede activar o

desactivar esta característica en la página de configuración del grupo seleccionando la pestaña Funciones de Lambda y seleccionando el detector IP. La detección automática de IP está activada si se selecciona Detectar y anular automáticamente los puntos de conexión del agente MQTT.

Para gestionar la detección automática con la API AWS IoT Greengrass, debe configurar la función de Lambda `IPDetector` del sistema. El siguiente procedimiento muestra cómo utilizar el comando create-function-definition-versionCLI para configurar la detección automática del núcleo de Greengrass.

1. Obtenga los ID de la versión de grupo y grupo de Greengrass de destino. En este procedimiento, suponemos que estos son el último grupo y la última versión de grupo. La siguiente consulta devuelve el grupo creado más recientemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   También puede hacer la consulta por nombre. No es necesario que los nombres de grupo sean únicos, por lo que podrían devolverse varios grupos.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

2. Copie los valores `Id` y `LatestVersion` del grupo de destino en la salida.
3. Obtenga la última versión del grupo.

   - Reemplace *`id-grupo`* con el `Id` que ha copiado.
   - Reemplace *`latest-group-version-id`* por el `LatestVersion` que ha copiado.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id latest-group-version-id
   ```

4. Desde el objeto `Definition` de la salida, copie el `CoreDefinitionVersionArn` y los ARN de todos los demás componentes de grupo excepto `FunctionDefinitionVersionArn`. Utilizará estos valores cuando cree una nueva versión del grupo.
5. En el objeto `FunctionDefinitionVersionArn` de la salida, copie el ID y la versión de la definición de funciones:

   ```
   arn:aws:greengrass:region:account-id:/greengrass/groups/function-definition-id/
   versions/function-definition-version-id
   ```

   > **Note**
   >
   > Si lo desea, para crear una definición de la función ejecute el comando `create-function-definition` y luego copie el ID del resultado.

6. Utilice el comando `get-function-definition-version` para obtener la definición de estado actual. Utilice el *`function-definition-id`*que ha copiado para la definición de la función. Por ejemplo, *`4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3`*.

   ```
   aws greengrass get-function-definition-version
   --function-definition-id function-definition-id
   --function-definition-version-id function-definition-version-id
   ```

   Anote las configuraciones de las funciones que aparecen. Tendrá que incluirlas cuando cree una nueva versión de la definición de funciones para evitar que se pierda la configuración de la definición actual.
7. Añada una versión de definición de la función a la definición de la característica.

   - *`function-definition-id`*Sustitúyalo por el `Id` que copió para la definición de la función. Por ejemplo, *`4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3`*.
   - *`arbitrary-function-id`*Sustitúyalo por un nombre para la función, por ejemplo**`auto-detection-function`**.
   - Añada en la matriz `functions` todas las funciones de Lambda que desee incluir en esta versión; por ejemplo, las funciones mostradas en el paso anterior.

   ```
   aws greengrass create-function-definition-version \
   ```

```
--function-definition-id function-definition-id \
--functions
 '[{"FunctionArn":"arn:aws:lambda:::function:GGIPDetector:1","Id":"arbitrary-
function-id","FunctionConfiguration":
{"Pinned":true,"MemorySize":32768,"Timeout":3}}]'\
--region us-west-2
```

8. Copie el `Arn` de la versión de definición de la característica del resultado.
9. Cree una versión de grupo que contenga la función de Lambda del sistema.

   - Reemplace *id-grupo* con el `Id` del grupo.
   - *core-definition-version-arn*Sustitúyalo por el `CoreDefinitionVersionArn` que copió de la última versión del grupo.
   - *function-definition-version-arn*Sustitúyala por la `Arn` que copió para la nueva versión de definición de funciones.
   - Reemplace los ARN para otros componentes del grupo (por ejemplo, `SubscriptionDefinitionVersionArn` o `DeviceDefinitionVersionArn`) que ha copiado de la última versión del grupo.
   - Elimine los parámetros no utilizados. Por ejemplo, elimine `--resource-definition-version-arn` si la versión de su grupo no contiene ningún recurso.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

10. Copie la `Version` del resultado. Este es el ID de la nueva versión del grupo.
11. Implemente el grupo con la nueva versión del grupo.

    - Reemplace *id-grupo* con el `Id` que ha copiado para el grupo.
    - *group-version-id*Sustitúyalo por el `Version` que copió para la nueva versión de grupo.

```
aws greengrass create-deployment \
```

```
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

Si desea introducir manualmente la dirección IP del núcleo de Greengrass, puede realizar este tutorial con una definición de funciones diferente y no incluir la función `IPDetector`. Esto impedirá que la función de detección pueda localizar e introducir automáticamente la dirección IP del núcleo de Greengrass.

Esta función de Lambda del sistema no es visible en la consola de Lambda. Una vez que la función se agregue a la última versión del grupo, se incluye en las implementaciones que se realizan desde la consola, a menos que se utilice la API para sustituirla o eliminarla.

Configurar el sistema Init para que inicie el daemon de Greengrass

Es una buena práctica configurar el sistema init para que inicie el daemon de Greengrass durante el arranque, especialmente si administra grandes flotas de dispositivos.

> **Note**
>
> Si ha utilizado `apt` para instalar el software AWS IoT Greengrass Core, puede utilizar los scripts systemd para habilitar el inicio en el arranque. Para obtener más información, consulte the section called “Uso de scripts de systemd para administrar el ciclo de vida del demonio de Greengrass”.

Existen diferentes tipos de sistema init, como initd, systemd y SystemV, que utilizan parámetros de configuración similares. En el siguiente ejemplo se muestra un archivo de servicio para systemd. El parámetro `Type` se establece en `forking` porque greengrassd (que se utiliza para iniciar Greengrass) bifurca el proceso del daemon de Greengrass y el parámetro `Restart` está establecido en `on-failure` para indicar a systemd que reinicie Greengrass si Greengrass entra en un estado de error.

> **Note**
>
> Para ver si su dispositivo usa systemd, ejecute el script `check_ggc_dependencies` como se describe en el Módulo 1. A continuación, para utilizar systemd, asegúrese de que el parámetro `useSystemd` en `config.json` está establecido en `yes`.

```
[Unit]
Description=Greengrass Daemon

[Service]
Type=forking
PIDFile=/var/run/greengrassd.pid
Restart=on-failure
ExecStart=/greengrass/ggc/core/greengrassd start
ExecReload=/greengrass/ggc/core/greengrassd restart
ExecStop=/greengrass/ggc/core/greengrassd stop

[Install]
WantedBy=multi-user.target
```

Véase también

- ¿Qué es AWS IoT Greengrass?
- the section called “Plataformas compatibles y requisitos”
- Empezando con AWS IoT Greengrass
- the section called “Información general sobre el modelo de objetos de grupo”
- the section called “Integración de la seguridad de hardware”

AWS IoT Greengrass Version 1política de mantenimiento

Utilice esta política de mantenimiento de AWS IoT Greengrass V1 para comprender los diferentes niveles de mantenimiento y actualizaciones del servicio de AWS IoT Greengrass V1 y del software versión AWS IoT Greengrass Core v1.x.

Temas

AWS IoT Greengrass esquema de control de versiones

AWS IoT Greengrass utiliza el control de versiones semántico para el AWS IoT Greengrass software Core. Las versiones semánticas siguen un sistema de números de principal.secundario.parche. La versión principal se incrementa para los cambios funcionales y de API que no sean compatibles con las versiones principales anteriores. La versión secundaria se incrementa en el caso de las versiones que añaden una nueva funcionalidad compatibles con versiones anteriores. La versión del parche se incrementa para incluir parches de seguridad o correcciones de errores. Desde su primera versión principal, la 1.0.0, AWS IoT Greengrass ha lanzado 11 versiones secundarias del software AWS IoT Greengrass Core versión1.x, siendo la 1.11.6 la última versión. Se recomienda que actualice el software AWS IoT Greengrass Core a la última versión disponible para aprovechar las nuevas características, mejoras y correcciones de errores.

En diciembre de 2020, AWS IoT Greengrass lanzó su primera actualización importante de la versión. Esta actualización incluía el servicio AWS IoT Greengrass V2 y la versión 2.0.3 del software AWS IoT Greengrass Core. Para las aplicaciones nuevas, le recomendamos encarecidamente que utilice AWS IoT Greengrass Version 2 y el software AWS IoT Greengrass Core v2.x. La versión 2 incluye nuevas características, incluye todas las características clave de la versión 1 y es compatible con

plataformas adicionales e implementaciones continuas en grandes flotas de dispositivos. Para obtener más información, consulte ¿Qué es AWS IoT Greengrass V2?.

Fases del ciclo de vida de las principales versiones del software AWS IoT Greengrass Core

Cada versión principal del software AWS IoT Greengrass Core tiene las siguientes tres fases secuenciales del ciclo de vida. Cada fase del ciclo de vida proporciona diferentes niveles de mantenimiento durante un período de tiempo posterior a la fecha de lanzamiento inicial.

- Fase de lanzamiento: AWS IoT Greengrass es posible que se lancen las siguientes actualizaciones:
  - Actualizaciones de versiones secundarias que proporcionan nuevas características o mejoras a las características existentes
  - Actualizaciones de la versión del parche que proporcionan parches de seguridad y correcciones de errores
- Fase de mantenimiento: AWS IoT Greengrass puede lanzar actualizaciones de la versión del parche que proporcionen parches de seguridad y correcciones de errores. AWS IoT Greengrass no lanzará nuevas características ni mejoras a las funciones existentes durante la fase de mantenimiento.
- Fase de vida útil prolongada:AWS IoT Greengrass no lanzará actualizaciones que proporcionen características, mejoras de las características existentes, parches de seguridad o correcciones de errores. Sin embargo, los puntos finales de Nube de AWS y las operaciones de la API seguirán disponibles y funcionarán de acuerdo con el AWS IoT GreengrassContrato de nivel de servicio. Los dispositivos que ejecutan la versión AWS IoT Greengrass Core 1.x del software pueden seguir conectados a la Nube de AWS y funcionando.

Cuando finalice la fase de vida útil prolongada de una versión principal de AWS IoT Greengrass, los puntos de conexión de Nube de AWS y las operaciones de la API quedarán obsoletas y dejarán de estar disponibles. Los dispositivos que ejecutan la versión AWS IoT Greengrass Core 1.x del software no podrán conectarse a los servicios Nube de AWS para funcionar.

Política de mantenimiento para el software AWS IoT Greengrass Core

La versión AWS IoT Greengrass Core 1.x del software entró en la fase de vida útil prolongada el 30 de junio de 2023. Después de esta fecha, la versión AWS IoT Greengrass Core 1.x del software permanecerá en la fase de vida útil prolongada hasta nuevo aviso.

La versión AWS IoT Greengrass Core 2.x del software se encuentra actualmente en la fase de lanzamiento y permanecerá en esa fase hasta nuevo aviso. AWS IoT Greengrasssigue añadiendo nuevas funciones y mejoras a la versión AWS IoT Greengrass Core 2.x del software . Por ejemplo, AWS IoT Greengrass lanzó el soporte para Windows en la versión 2.5.0 del software AWS IoT Greengrass Core. AWS IoT Greengrass lanza parches de seguridad y correcciones de errores para todas las versiones secundarias de AWS IoT Greengrass Core v2.x durante al menos 1 año después de la fecha de lanzamiento. Para obtener más información, consulte What's New in AWS IoT Greengrass V2.

Programación de la fase de mantenimiento

El 30 de junio de 2023 finalizó la fase de mantenimiento de la versión 1.11.x del software AWS IoT Greengrass Core. El 31 de marzo de 2022, finalizó la fase de mantenimiento del software AWS IoT Greengrass Core versión 1.10.x. La fase de mantenimiento finaliza para algunos artefactos y características del software AWS IoT Greengrass Core versión 1.x antes de estas fechas. Para obtener más información, consulte Fin del programa de mantenimiento.

Si tiene un plan AWS Support, la fase de mantenimiento de la versión 1.x del software AWS IoT Greengrass Core no afecta a su plan AWS Support. Puede seguir abriendo los tickets AWS Support incluso después de que finalice la fase de mantenimiento. Si tiene alguna pregunta o duda, póngase en contacto AWS Support con su contacto o haga una pregunta en AWSre:Post usando la etiqueta AWS IoT Greengrass.

Calendario de obsolescencia

Actualmente, no hay ningún plan para dejar de dar soporte a la versión 1.x del software AWS IoT Greengrass Core. Los puntos de conexión AWS IoT Greengrass V1 y las operaciones de la API permanecerán disponibles hasta nuevo aviso. La versión AWS IoT Greengrass Core 1.11.6 del software entró en la fase de vida útil prolongada el 30 de junio de 2023. Durante esta fase, los dispositivos que ejecutan la versión 1.x del software AWS IoT Greengrass Core pueden seguir conectándose al servicio AWS IoT Greengrass V1 para funcionar hasta nuevo aviso.

Si AWS IoT Greengrass V1 deja de recibir soporte en el futuro, AWS IoT Greengrass avisará con 12 meses de antelación antes de que esto suceda. Esto le ayudará a planificar la actualización de sus aplicaciones que vaya a utilizar AWS IoT Greengrass V2 y del software AWS IoT Greengrass Core v2.x. Para obtener más información sobre cómo actualizar sus aplicaciones a V2, consulte Pasar de AWS IoT Greengrass V1 a la versión 2.

Política de soporte para las funciones de Lambda AWS Lambdade los dispositivos principales de Greengrass

AWS IoT Greengrass le permite ejecutar funciones Lambda AWS Lambdaen dispositivos de IoT. AWS Lambda proporciona una política de soporte y plazos que determinan el soporte para los tiempos de ejecución de Lambda en AWS IoT Greengrass. Cuando un tiempo de ejecución de Lambda llega al final de la fase de soporte, AWS IoT Greengrass también finaliza el soporte para ese tiempo de ejecución. Para obtener más información, consulte la Política de soporte en tiempo de ejecución en la Guía de desarrolladores de AWS Lambda.

Cuando un tiempo de ejecución de Lambda llega al final del soporte, no se pueden crear ni actualizar funciones de Lambda que usen ese tiempo de ejecución. Sin embargo, puede seguir implementando estas funciones de Lambda en los dispositivos principales de Greengrass e invocar las funciones de Lambda implementadas. Esta política también se aplica a AWS IoT Greengrass V2.

Política de compatibilidad de AWS IoT Device Tester para AWS IoT Greengrass V1

AWS IoTEl comprobador de dispositivos (Device Tester, IDT) AWS IoT Greengrass V1 le permite validar y calificar sus AWS IoT Greengrass dispositivos para su inclusión en el AWS PartnerCatálogo de dispositivos. A partir del 4 de abril de 2022, AWS IoT el comprobador de dispositivos (IDT) para AWS IoT Greengrass V1 ya no genera informes de calificación firmados. Ya no puede incluir nuevos dispositivos AWS IoT Greengrass V1 en el Catálogo de dispositivos AWS Partner a través del AWSPrograma de calificación de dispositivos. Si bien no puede calificar los dispositivos Greengrass V1, puede seguir usando IDT para AWS IoT Greengrass V1 para probar sus dispositivos Greengrass V1. Le recomendamos que utilice IDT para AWS IoT Greengrass V2 para calificar y publicar los dispositivos Greengrass en el AWS PartnerCatálogo de dispositivos. Para obtener más información, consulte Política de compatibilidad de AWS IoT Device Tester para AWS IoT Greengrass V1.

Fin del programa de mantenimiento

En la siguiente tabla se enumeran las fechas de finalización del mantenimiento de los artefactos y las características de la versión AWS IoT Greengrass Core v1.x. Si tiene preguntas sobre el programa o la política de mantenimiento, póngase en contacto con Soporte de AWS.

Artefacto o característica	Fecha de fin del mantenimiento
Instalación del repositorio APT de Greengrass	11 de febrero de 2022
Conector de la clasificación de ML Image	31 de marzo de 2022
Conector de detección de ML Object	31 de marzo de 2022
Conector de ML Feedback	31 de marzo de 2022
Conector de AWS IoT Analytics	31 de marzo de 2022
Conector de notificaciones Twilio	31 de marzo de 2022
Conector de integración Splunk	31 de marzo de 2022
Conector Serial Stream	31 de marzo de 2022
Conector de integración MetricBase de ServiceNow	31 de marzo de 2022
Conector Raspberry Pi GPIO	31 de marzo de 2022
Software AWS IoT Greengrass Core versión 1.10.x	31 de marzo de 2022
Imágenes de Docker del software AWS IoT Greengrass Core v1.x	30 de junio de 2022
Software AWS IoT Greengrass Core versión 1.11.x	30 de junio de 2023
Software AWS IoT Greengrass Core versión 1.11.x Snap	31 de diciembre de 2023

Fin del mantenimiento de las imágenes de Docker del software AWS IoT Greengrass Core versión 1.x

El 30 de junio de 2022, AWS IoT Greengrass finalizó el mantenimiento de las imágenes de Docker del software AWS IoT Greengrass Core versión 1.x publicadas en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub. Puede seguir descargando estas imágenes de Docker desde Amazon ECR y Docker Hub hasta el 30 de junio de 2023, es decir, un año después de que finalice el mantenimiento. Sin embargo, las imágenes de Docker de la versión 1.x del software AWS IoT Greengrass Core ya no reciben parches de seguridad ni correcciones de errores una vez finalizado el mantenimiento el 30 de junio de 2022. Si ejecuta una carga de trabajo de producción que depende de estas imágenes de Docker, le recomendamos que cree sus propias imágenes de Docker con los archivos Docker que AWS IoT Greengrass proporciona. Para obtener más información, consulte AWS IoT Greengrass Software Docker.

Fin del mantenimiento del repositorio APT de la versión 1.x del software AWS IoT Greengrass Core

El 11 de febrero de 2022, AWS IoT Greengrass finalizó el mantenimiento de la opción de instalar el software AWS IoT Greengrass Core versión 1.x desde un repositorio de APT. El repositorio de APT se eliminó en esta fecha, por lo que ya no se puede utilizar el repositorio de APT para actualizar el software AWS IoT Greengrass Core o instalar el software AWS IoT Greengrass Core en dispositivos nuevos. En los dispositivos en los que haya agregado el repositorio AWS IoT Greengrass, debe eliminarlo de la lista de fuentes. Le recomendamos que actualice la versión 1.x del software AWS IoT Greengrass Core mediante archivos tar.

Fin del mantenimiento del repositorio APT de la versión 1.11.x Snap del software AWS IoT Greengrass Core

El 31 de diciembre de 2023 AWS IoT Greengrass finalizará el mantenimiento de la versión 1.11.x Snap del software AWS IoT Greengrass principal, publicada en snapcraft.io. Los dispositivos que ejecutan actualmente el Snap seguirán funcionando hasta nuevo aviso. Sin embargo, el Snap AWS IoT Greengrass principal ya no recibirá parches de seguridad ni correcciones de errores después que finalice el mantenimiento.

Empezar con AWS IoT Greengrass

Este tutorial de introducción incluye varios módulos diseñados para mostrarle AWS IoT Greengrass los conceptos básicos y ayudarle a empezar a usarlos AWS IoT Greengrass. Este tutorial cubre conceptos fundamentales, como:

- Configuración de AWS IoT Greengrass núcleos y grupos.
- El proceso de despliegue para ejecutar AWS Lambda funciones en la periferia.
- Conectar AWS IoT dispositivos, denominados dispositivos cliente, al AWS IoT Greengrass núcleo.
- Crear suscripciones para permitir la comunicación MQTT entre las funciones locales de Lambda, los dispositivos cliente y. AWS IoT

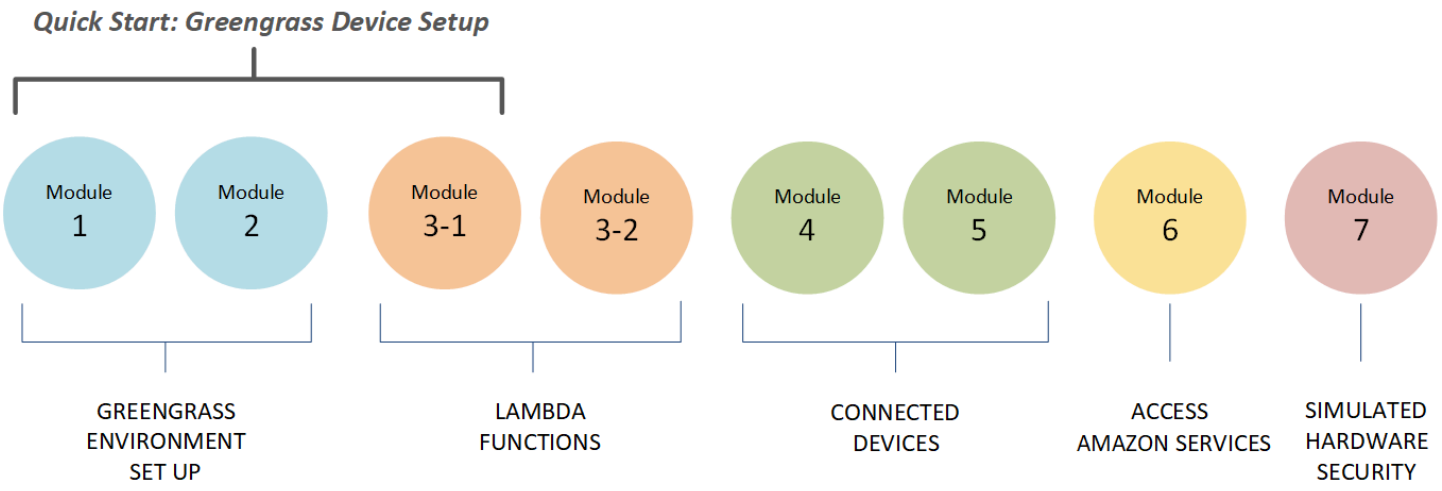
Elija cómo empezar AWS IoT Greengrass

Puede elegir cómo utilizar este tutorial para configurar el dispositivo principal:

- Ejecute la configuración del dispositivo Greengrass en su dispositivo principal, lo que le llevará desde instalar AWS IoT Greengrass dependencias hasta probar una función Lambda de Hello World en cuestión de minutos. Este script reproduce los pasos del Módulo 1 al Módulo 3-1.

  - o bien -

- Siga los pasos del Módulo 1 al Módulo 3-1 para examinar más de cerca los requisitos y procesos de Greengrass. En estos pasos se configura el dispositivo principal, se crea y configura un grupo de Greengrass que contenga una función de Lambda Hello World y se implementa el grupo de Greengrass. Normalmente, esto tarda una o dos horas en completarse.



Quick Start (Inicio rápido)

La configuración del dispositivo Greengrass configura el dispositivo principal y los recursos de Greengrass. El script:

- Instala las dependencias. AWS IoT Greengrass
- Descarga el certificado de CA raíz y el certificado y las claves del dispositivo principal.
- Descarga, instala y configura el software AWS IoT Greengrass principal del dispositivo.
- Inicia el proceso del daemon de Greengrass en el dispositivo principal.
- Crea o actualiza el rol de servicio de Greengrass, si es necesario.
- Crea un grupo de Greengrass y un núcleo de Greengrass.
- (Opcional) Crea una función de Lambda Hello World, una suscripción y una configuración de registro local.
- (Opcional) Implementa el grupo de Greengrass.

Módulos 1 y 2

En el Módulo 1 y el Módulo 2 se describe cómo configurar el entorno. (O bien utilice la configuración del dispositivo Greengrass para que estos módulos se ejecuten automáticamente).

- Configure Greengrass en el dispositivo principal.
- Ejecute el script comprobador de dependencias.
- Cree un grupo de Greengrass y un núcleo de Greengrass.
- Descargue e instale el software AWS IoT Greengrass Core más reciente desde un archivo tar.gz.

- Inicie el proceso del daemon de Greengrass en el núcleo.

> **Note**
>
> AWS IoT Greengrass también ofrece otras opciones para instalar el software AWS IoT Greengrass Core, incluidas `apt` las instalaciones en plataformas Debian compatibles. Para obtener más información, consulte the section called “Instalación del software AWS IoT Greengrass Core”.

Módulos 3-1 y 3-2

En el Módulo 3-1 y el Módulo 3-2 se describe cómo utilizar las funciones de Lambda locales. (O bien utilice la configuración del dispositivo Greengrass para que el Módulo 3-1 se ejecute automáticamente).

- Cree funciones Lambda de Hello World en. AWS Lambda
- Añada funciones de Lambda al grupo de Greengrass.
- Cree suscripciones que permitan la comunicación MQTT entre las funciones de Lambda y. AWS IoT
- Configure el registro local en los componentes del sistema de Greengrass y las funciones de Lambda.
- Implemente un grupo de Greengrass que contenga las funciones de Lambda y las suscripciones.
- Envíe mensajes desde funciones de Lambda locales a. AWS IoT
- Invoque funciones Lambda locales desde. AWS IoT
- Pruebe las funciones bajo demanda y de larga duración.

Módulos 4 y 5

En el Módulo 4 se muestra cómo los dispositivos de cliente se conectan al núcleo y se comunican entre sí.

En el Módulo 5 se muestra cómo los dispositivos cliente pueden utilizar sombras para controlar el estado.

- Registre y aprovisione AWS IoT los dispositivos (representados por terminales de línea de comandos).

- Instale el SDK para dispositivos con AWS IoT para Python. Los dispositivos cliente lo utilizan para detectar el núcleo de Greengrass.
- Añada los dispositivos de cliente a su grupo de Greengrass.
- Cree suscripciones que permitan la comunicación MQTT.
- Implemente un grupo de Greengrass que contenga los dispositivos cliente.
- Pruebe device-to-device la comunicación.
- Pruebe las actualizaciones de estado de sombra.

Módulo 6

En el Módulo 6 se muestra cómo las funciones de Lambda pueden acceder a la Nube de AWS.

- Cree un rol de grupo de Greengrass que permita el acceso a los recursos de Amazon DynamoDB.
- Añada una función de Lambda al grupo de Greengrass. Esta función usa el AWS SDK de Python para interactuar con DynamoDB.
- Cree suscripciones que permitan la comunicación MQTT.
- Pruebe la interacción con Amazon DynamoDB.

Módulo 7

En el Módulo 7 se muestra cómo configurar un módulo de seguridad de hardware (HSM) simulado para su uso con un núcleo de Greengrass.

> **Important**
>
> Este módulo avanzado solo se ofrece para la experimentación y las pruebas iniciales. No está destinado a su uso con fines de producción de ningún tipo.

- Instale y configure un HSM basado en software y una clave privada.
- Configure el núcleo de Greengrass para utilizar la seguridad del hardware.
- Pruebe la configuración de seguridad del hardware.

Requisitos

Necesitará lo siguiente para completar este tutorial:

- Un sistema Mac, Windows o de tipo UNIX.
- Un. Cuenta de AWS Si no dispone de una, consulte the section called “Crea un Cuenta de AWS”.
- El uso de una AWS región que admite AWS IoT Greengrass. Para ver la lista de regiones compatibles AWS IoT Greengrass, consulte los AWS puntos finales y las cuotas en. Referencia general de AWS

> **Note**
>
> Tome nota de las suyas Región de AWS y asegúrese de que se utilice de forma coherente a lo largo de este tutorial. Si cambias de dirección Región de AWS durante el tutorial, es posible que tengas problemas para completar los pasos.

- Un Raspberry Pi 4 modelo B o un Raspberry Pi 3 modelo B/B+, con una tarjeta microSD de 8 GB o una instancia Amazon EC2. Puesto que lo ideal es usar AWS IoT Greengrass con hardware físico, le recomendamos que utilice un Raspberry Pi.

> **Note**
>
> Ejecute el comando siguiente para obtener el modelo de su Raspberry Pi:
>
> ```
> cat /proc/cpuinfo
> ```
>
> Cerca de la parte inferior de la lista, anote el valor del atributo `Revision` y, a continuación, consulte la tabla ¿Qué Pi tengo?. Por ejemplo, si el valor de `Revision` es `a02082`, la tabla muestra que el modelo de Pi es un 3 Model B.
> Ejecute el siguiente comando para determinar la arquitectura de su Raspberry Pi:
>
> ```
> uname -m
> ```
>
> En este tutorial, el resultado debe ser mayor o igual que `armv71`.

- Conocimientos básicos sobre Python.

Aunque este tutorial está diseñado para ejecutarse AWS IoT Greengrass en una Raspberry Pi, AWS IoT Greengrass también es compatible con otras plataformas. Para obtener más información, consulte the section called “Plataformas compatibles y requisitos”.

Crea un Cuenta de AWS

Si no tienes una Cuenta de AWS, sigue estos pasos para crear y activar una Cuenta de AWS:

Inscríbete en una Cuenta de AWS

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

Para suscribirse a una Cuenta de AWS

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

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

   Cuando te registras en una Cuenta de AWS, Usuario raíz de la cuenta de AWSse crea una. 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.

AWS te envía un correo electrónico de confirmación una vez finalizado el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en https://aws.amazon.com/ y seleccionando Mi cuenta.

Creación de un usuario con acceso administrativo

Después de crear un usuario administrativo Cuenta de AWS, asegúrelo Usuario raíz de la cuenta de AWS AWS IAM Identity Center, habilite y cree un usuario administrativo para no usar el usuario root en las tareas diarias.

Proteja su Usuario raíz de la cuenta de AWS

1. Inicie sesión AWS Management Consolecomo propietario de la cuenta seleccionando el usuario root e introduciendo su dirección de Cuenta de AWS correo electrónico. En la siguiente página, escriba su contraseña.

   Para obtener ayuda para iniciar sesión con el usuario raíz, consulte Signing in as the root user en la Guía del usuario de AWS Sign-In .

2. Active la autenticación multifactor (MFA) para el usuario raíz.

   Para obtener instrucciones, consulte Habilitar un dispositivo MFA virtual para el usuario Cuenta de AWS raíz (consola) en la Guía del usuario de IAM.

Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

   Consulte las instrucciones en Activar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center .

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

   Para ver un tutorial sobre su uso Directorio de IAM Identity Center como fuente de identidad, consulte Configurar el acceso de los usuarios con la configuración predeterminada Directorio de IAM Identity Center en la Guía del AWS IAM Identity Center usuario.

Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

  Para obtener ayuda para iniciar sesión con un usuario del Centro de identidades de IAM, consulte Iniciar sesión en el portal de AWS acceso en la Guía del AWS Sign-In usuario.

Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

   Para conocer las instrucciones, consulte Create a permission set en la Guía del usuario de AWS IAM Identity Center .

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

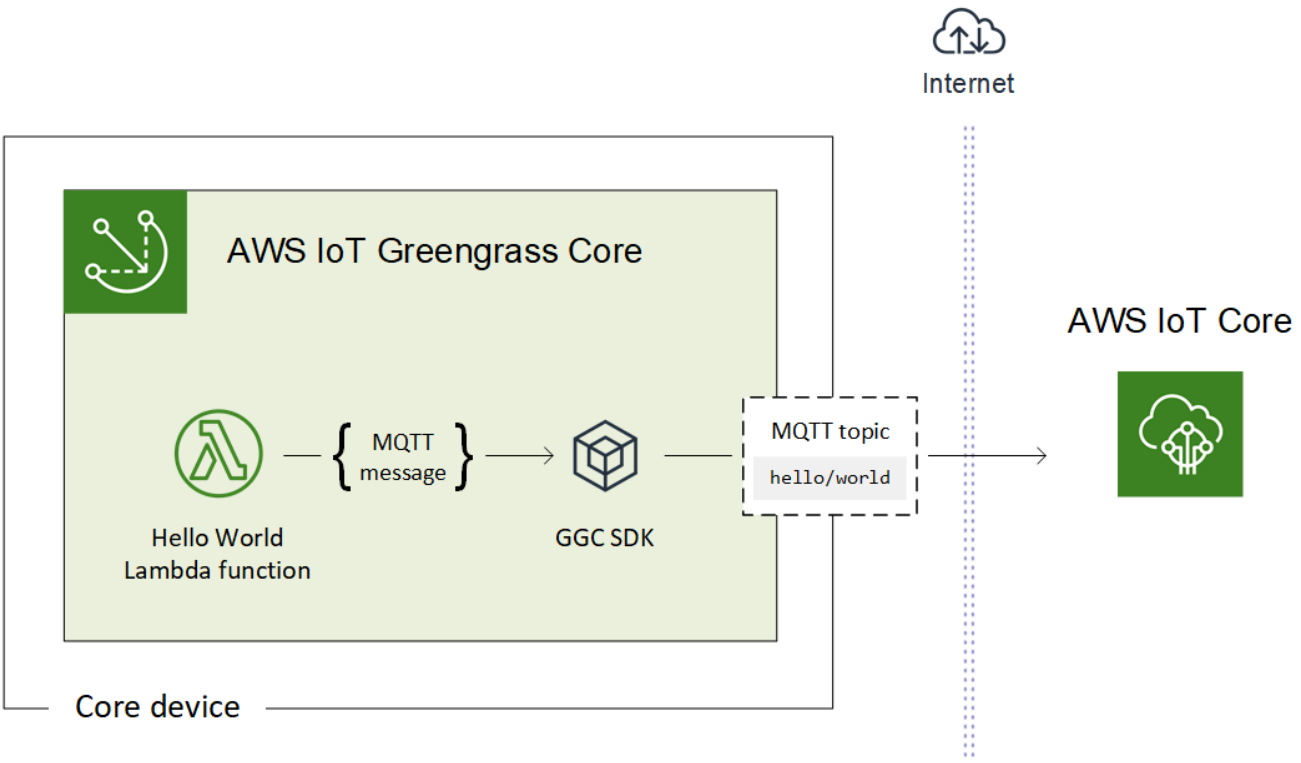
   Para conocer las instrucciones, consulte Add groups en la Guía del usuario de AWS IAM Identity Center .

> **Important**
>
> En este tutorial, suponemos que la cuenta de usuario de IAM tiene permisos de acceso de administrador.

Inicio rápido: configuración del dispositivo Greengrass

La configuración del dispositivo de Greengrass es un script que configura el dispositivo principal en cuestión de minutos, para que pueda comenzar a utilizar AWS IoT Greengrass. Utilice este script para:

1. Configurar el dispositivo e instala el software AWS IoT Greengrass Core.
2. Configura los recursos basados en la nube.
3. Implementa un grupo de Greengrass con una función Hello World de Lambda que envía mensajes MQTT a AWS IoT Greengrass desde el núcleo de AWS IoT. Este paso se configura el entorno de Greengrass que se muestra en el siguiente diagrama.



Requisitos

La configuración del dispositivo Greengrass presenta los siguientes requisitos:

- El dispositivo principal debe utilizar una plataforma compatible. El dispositivo debe tener instalado un administrador de paquetes adecuado: `apt`, `yum` o `opkg`.
- El usuario de Linux que ejecuta el script debe tener permisos para ejecutar como `sudo`.
- Debe propocionar sus credenciales de Cuenta de AWS. Para obtener más información, consulte the section called “Proporcione credenciales de Cuenta de AWS”.

> **Note**
>
> La configuración del dispositivo Greengrass instala la versión más reciente del software AWS IoT Greengrass Core en el dispositivo. Al instalar el software AWS IoT Greengrass Core, acepta el acuerdo de licencia del software de Greengrass Core.

Ejecución de la configuración del dispositivo Greengrass

Puede ejecutar la configuración del dispositivo Greengrass en unos pocos pasos. Después de proporcionar las credenciales de su Cuenta de AWS, el script aprovisiona el dispositivo principal de Greengrass e implementa un grupo de Greengrass en cuestión de minutos. Ejecute los siguientes comandos en una ventana de terminal del dispositivo de destino.

> **Note**
>
> Estos pasos muestran cómo ejecutar la secuencia de comandos en modo interactivo, que le pide que introduzca o acepte cada valor de entrada. Para obtener información sobre cómo ejecutar el script de forma silenciosa, consulte the section called “Ejecución de la configuración del dispositivo Greengrass en modo silencioso”.

1. Proporcione las credenciales. En este procedimiento, suponemos que proporciona credenciales de seguridad temporales como variables de entorno.

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

> **Note**
>
> Si ejecuta la configuración del dispositivo Greengrass en una plataforma Raspbian u OpenWrt, copie estos comandos. Debe proporcionarlos de nuevo después de reiniciar el dispositivo.

2. Descargue e inicie el script. Puede utilizar `wget` o `curl` para descargar el script.

   `wget`:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

   `curl`:

```
curl https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh > gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

3. Avance por los símbolos del sistema para obtener los valores de entrada. Puede pulsar la tecla Enter (Intro) para utilizar el valor predeterminado o escribir un valor personalizado y, a continuación, pulsar Enter (Intro).

   El script escribe mensajes de estado en el terminal similares a los siguientes.

```
############## Greengrass Device Setup v1.0.0 ##############
[GreengrassDeviceSetup] The Greengrass Device Setup bootstrap log is available at: /tmp/greengrass-device-setup-bootstrap-1575933831.log
[GreengrassDeviceSetup] Using package management tool: yum...
[GreengrassDeviceSetup] Using runtime: python3.7...
[GreengrassDeviceSetup] Installing a dedicated pip for Greengrass Device Setup...
[GreengrassDeviceSetup] Validating and installing required dependencies...
[GreengrassDeviceSetup] The Greengrass Device Setup configuration is complete. Starting the Greengrass environment setup...
[GreengrassDeviceSetup] Forwarding command-line parameters: bootstrap-greengrass-interactive

[GreengrassDeviceSetup] Validating the device environment...
[GreengrassDeviceSetup] Validation of the device environment is complete.

[GreengrassDeviceSetup] Running the Greengrass environment setup...
[GreengrassDeviceSetup] The Greengrass environment setup is complete.

[GreengrassDeviceSetup] Configuring cloud-based Greengrass group management...
[GreengrassDeviceSetup] The Greengrass group configuration is complete.

[GreengrassDeviceSetup] Preparing the Greengrass core software...
[GreengrassDeviceSetup] The Greengrass core software is running.

[GreengrassDeviceSetup] Configuring the group deployment...
[GreengrassDeviceSetup] The group deployment is complete.
```

4. Si el dispositivo principal ejecuta Raspbian u OpenWrt, reinicie el dispositivo cuando se le solicite, proporcione las credenciales y, a continuación, reinicie el script.

   a. Cuando se le solicite que reinicie el dispositivo, ejecute uno de los siguientes comandos.

      Para plataformas Raspbian:

      ```
      sudo reboot
      ```

      Para plataformas OpenWrt:

      ```
      reboot
      ```

   b. Una vez que se reinicie el dispositivo, abra el terminal y proporcione las credenciales como variables de entorno.

      ```
      export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
      export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
      export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
      ```

   c. Reinicie el script.

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
      ```

   d. Cuando se le pregunte si desea utilizar los valores de entrada de la sesión anterior o iniciar una nueva instalación, escriba `yes` para volver a utilizar los valores de entrada.

> **Note**
>
> En las plataformas que soliciten que se reinicie, los valores de entrada de la sesión anterior, salvo las credenciales, se almacenan temporalmente en el archivo `GreengrassDeviceSetup.config.info`.

Una vez finalizada la configuración, el terminal muestra un mensaje de estado de operación correcta similar al siguiente.

```
================================================================================

Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.


Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/versi
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.

================================================================================
```

5. Revise el nuevo grupo de Greengrass que el script configura con los valores de entrada que proporciona.

   a. Inicie sesión en la AWS Management Console en su equipo y abra la consola AWS IoT.

   > **Note**
   >
   > Asegúrese de que la Región de AWS seleccionada en la consola sea la misma que utilizó para configurar el entorno de Greengrass. De forma predeterminada, la región es oeste de EE. UU. (Oregón).

   b. En el panel de navegación, expanda Dispositivos de Greengrass y, a continuación, elija Grupos (V1) para localizar el grupo recién creado.

6. Si ha incluido la función Hello World de Lambda, la configuración del dispositivo Greengrass implementa el grupo de Greengrass en el dispositivo principal. Para probar la función de Lambda u obtener información acerca de cómo eliminar la función de the section called “Verificación de la ejecución de la función de Lambda en el dispositivo central” del grupo, continúe hacia en el Módulo 3-1 del tutorial de introducción.

   > **Note**
   >
   > Asegúrese de que la Región de AWS seleccionada en la consola sea la misma que utilizó para configurar el entorno de Greengrass. De forma predeterminada, la región es oeste de EE. UU. (Oregón).

   Si no ha incluido la función Hello World de Lambda, puede crear su propia función de Lambda o probar otras características de Greengrass. Por ejemplo, puede añadir el conector de implementación de aplicaciones Docker al grupo y utilizarlo para implementar contenedores de Docker en el dispositivo principal.

Solución de problemas con

Puede utilizar la siguiente información para solucionar problemas con la configuración del dispositivo AWS IoT Greengrass.

Error: no se encontró Python (python3.7). Intentando instalarlo...

Solución: es posible que aparezca este error al trabajar con una instancia de Amazon EC2. Este error se produce cuando Python no está instalado en la carpeta `/usr/bin/python3.7`. Para resolver este error, mueva Python al directorio correcto después de instalarlo:

```
sudo ln -s /usr/local/bin/python3.7 /usr/bin/python3.7
```

Solución de problemas adicionales

Para solucionar los problemas adicionales con la configuración del dispositivo AWS IoT Greengrass, puede buscar información de depuración en los archivos de registro:

- Para solucionar problemas de configuración del dispositivo Greengrass, compruebe el archivo `/tmp/greengrass-device-setup-bootstrap-epoch-timestamp.log`.
- Para solucionar problemas de configuración del entorno de núcleo o del grupo de Greengrass, compruebe el archivo `GreengrassDeviceSetup-date-time.log` en el mismo directorio que `gg-device-setup-latest.sh` o en la ubicación que haya especificado.

Si necesita más ayuda para solucionar problemas, consulte Solución de problemas o eche un vistazo a a pestaña AWS IoT Greengrass en AWS re:Post.

Opciones de configuración del dispositivo Greengrass

Complete la configuración del dispositivo Greengrass para acceder a los recursos de AWS y configurar el entorno de Greengrass.

Proporcione credenciales de Cuenta de AWS

La configuración del dispositivo Greengrass utiliza sus credenciales de Cuenta de AWS para acceder a los recursos de AWS. Admite credenciales a largo plazo para un usuario de IAM o credenciales de seguridad temporales de un rol de IAM.

Primero, obtenga las credenciales

- Para utilizar credenciales a largo plazo, proporcione el ID de clave de acceso y la clave de acceso secreta del usuario de IAM. Para obtener información acerca de cómo crear claves de acceso para credenciales a largo plazo, consulte Administración de las claves de acceso de los usuarios de IAM en la Guía de usuario de IAM.

- Para utilizar credenciales de seguridad temporales (opción recomendada), proporcione el ID de clave de acceso, la clave de acceso secreta y el token de sesión desde un rol de IAM asumido. Para obtener información acerca cómo extraer credenciales de seguridad temporales con el comando AWS STS de `assume-role`, consulte Uso de credenciales de seguridad temporales con AWS CLI en la Guía de usuario de IAM..

> **Note**
>
> A efectos de este tutorial, suponemos que el usuario de IAM o el rol de IAM tienen permisos de acceso de administrador.

A continuación, proporcione sus credenciales a la configuración del dispositivo Greengrass de dos maneras:

- Como variables de entorno. Establezca las variables de entorno `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` y `AWS_SESSION_TOKEN` (si es necesario) antes de iniciar el script, como se muestra en el paso 1 de the section called “Ejecución de la configuración del dispositivo Greengrass”.

- Como valores de entrada. Introduzca su ID de clave de acceso, clave de acceso secreta y valores de token de sesión (si es necesario) directamente en el terminal después de iniciar el script.

La configuración del dispositivo Greengrass no guarda ni almacena las credenciales.

Proporcionar los valores de entrada

En el modo interactivo, la configuración del dispositivo Greengrass solicita los valores de entrada. Puede pulsar la tecla Enter (Intro) para utilizar el valor predeterminado o escribir un valor personalizado y, a continuación, pulsar Enter (Intro). En modo silencioso, se proporcionan los valores de entrada después de iniciar el script.

Valores de entrada

ID de clave de acceso de AWS

El ID de clave de acceso de las credenciales de seguridad temporales o a largo plazo. Especifique esta opción como valor de entrada solo si no proporciona las credenciales como variables de entorno. Para obtener más información, consulte the section called “Proporcione credenciales de Cuenta de AWS”.

Nombre de la opción para el modo silencioso: `--aws-access-key-id`

Clave de acceso secreta de AWS

La clave de acceso secreta de las credenciales de seguridad temporales o a largo plazo. Especifique esta opción como valor de entrada solo si no proporciona las credenciales como variables de entorno. Para obtener más información, consulte the section called “Proporcione credenciales de Cuenta de AWS”.

Nombre de la opción para el modo silencioso: `--aws-secret-access-key`

Token de sesión de AWS

El token de sesión de las credenciales de seguridad temporales. Especifique esta opción como valor de entrada solo si no proporciona las credenciales como variables de entorno. Para obtener más información, consulte the section called “Proporcione credenciales de Cuenta de AWS”.

Nombre de la opción para el modo silencioso: `--aws-session-token`

Región de AWS

La Región de AWS en la que desea crear el grupo de Greengrass. Para obtener las lista de las Región de AWS compatibles, consulte AWS IoT Greengrass en la Referencia general de Amazon Web Services.

Valor predeterminado: `us-west-2`

Nombre de la opción para el modo silencioso: `--region`

Group name

El nombre del grupo de Greengrass.

Valor predeterminado: `GreengrassDeviceSetup_Group_`*`guid`*

Nombre de la opción para el modo silencioso: `--group-name`

Core name (Nombre principal)

El nombre del núcleo de Greengrass. El núcleo es un dispositivo (objeto) AWS IoT que ejecuta el software AWS IoT Greengrass Core. El núcleo se añade al registro de AWS IoT y al grupo de Greengrass. Si proporciona un nombre, este debe ser exclusivo en la Cuenta de AWS y la Región de AWS.

Valor predeterminado: `GreengrassDeviceSetup_Core_`*`guid`*

Nombre de la opción para el modo silencioso: `--core-name`

Ruta de instalación del software principal AWS IoT Greengrass

La ubicación del sistema de archivos del dispositivo donde desea instalar el software AWS IoT Greengrass Core.

Valor predeterminado: `/`

Nombre de la opción para el modo silencioso: `--ggc-root-path`

Hello World Lambda function (Función Hello World de Lambda)

Indica si se desea incluir una función Hello World de Lambda en el grupo de Greengrass. La función publica un mensaje MQTT en el tema `hello/world` cada cinco segundos.

El script crea y publica esta función de Lambda definida por el usuario en AWS Lambda y la añade al grupo de Greengrass. El script también crea una suscripción en el grupo que permite a la función enviar mensajes MQTT a AWS IoT.

> **Note**
>
> Esta es una función de Lambda de Python 3.7. Si no está instalado Python 3.7 en el dispositivo y el script no puede instalarlo, este imprime un mensaje de error en el terminal. Para incluir la función de Lambda en el grupo, debe instalar Python 3.7 manualmente y reiniciar el script. Para crear el grupo de Greengrass sin la función de Lambda, reinicie el script y escriba no cuando se le solicite que incluya la función.

Valor predeterminado: no

Nombre de la opción para el modo silencioso: `--hello-world-lambda`. Esta opción no toma un valor. Inclúyala en su comando si desea crear la función.

Deployment timeout (Tiempo de espera de implementación)

Número de segundos antes de que la configuración del dispositivo Greengrass deje de comprobar el estado de la implementación del grupo de Greengrass. Este se utiliza solo cuando el grupo incluye la función Hello World de Lambda. De lo contrario, el grupo no se implementa.

El tiempo de implementación depende de la velocidad de la red. Para velocidades de red lentas, puede aumentar este valor.

Valor predeterminado: `180`

Nombre de la opción para el modo silencioso: `--deployment-timeout`

Log path (Ruta de registro)

Ubicación del archivo de registro que contiene información acerca de las operaciones de configuración del grupo y del núcleo de Greengrass. Utilice este registro para solucionar problemas de implementación y otros problemas con la configuración del núcleo y del grupo de Greengrass.

Valor predeterminado: `./`

Nombre de la opción para el modo silencioso: `--log-path`

Verbosidad

Indica si se debe imprimir información de registro detallada en el terminal mientras se ejecuta el script. Puede utilizar esta información para solucionar problemas de configuración del dispositivo.

Valor predeterminado: no

Nombre de la opción para el modo silencioso: `--verbose`. Esta opción no toma un valor. Incluya en el comando si desea imprimir información detallada del registro.

Ejecución de la configuración del dispositivo Greengrass en modo silencioso

Puede ejecutar la configuración del dispositivo Greengrass en modo silencioso para que el script no le pida ningún valor. Para ejecutar en modo silencioso, especifique el modo `bootstrap-greengrass` y los valores de entrada después de iniciar el script. Puede omitir los valores de entrada si desea utilizar sus valores predeterminados.

El procedimiento depende de si proporciona las credenciales de Cuenta de AWS como variables de entorno antes de iniciar el script o como valores de entrada después de iniciar el script.

Proporcionar las credenciales como variables de entorno

1. Proporcione sus credenciales como variables de entorno. En el ejemplo siguiente se exportan credenciales temporales, que incluyen el token de sesión.

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

> **Note**
>
> Si ejecuta la configuración del dispositivo Greengrass en una plataforma Raspbian u OpenWrt, copie estos comandos. Debe proporcionarlos de nuevo después de reiniciar el dispositivo.

2. Descargue e inicie el script. Proporcione los valores de entrada según sea necesario. Por ejemplo:

   - Para utilizar todos los valores predeterminados:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   ```

   - Para especificar valores personalizados:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --region us-east-1
   --group-name Custom_Group_Name
   --core-name Custom_Core_Name
   --ggc-root-path /custom/ggc/root/path
   --deployment-timeout 300
   --log-path /customized/log/path
   --hello-world-lambda
   --verbose
   ```

> **Note**
>
> Para utilizar `curl` para descargar el script, reemplace `wget -q -O` con `curl` en el comando.

3. Si el dispositivo principal ejecuta Raspbian u OpenWrt, reinicie el dispositivo cuando se le solicite, proporcione las credenciales y, a continuación, reinicie el script.

a. Cuando se le solicite que reinicie el dispositivo, ejecute uno de los siguientes comandos.

   Para plataformas Raspbian:

   ```
   sudo reboot
   ```

   Para plataformas OpenWrt:

   ```
   reboot
   ```

b. Una vez que se reinicie el dispositivo, abra el terminal y proporcione las credenciales como variables de entorno.

   ```
   export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
   export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
   ```

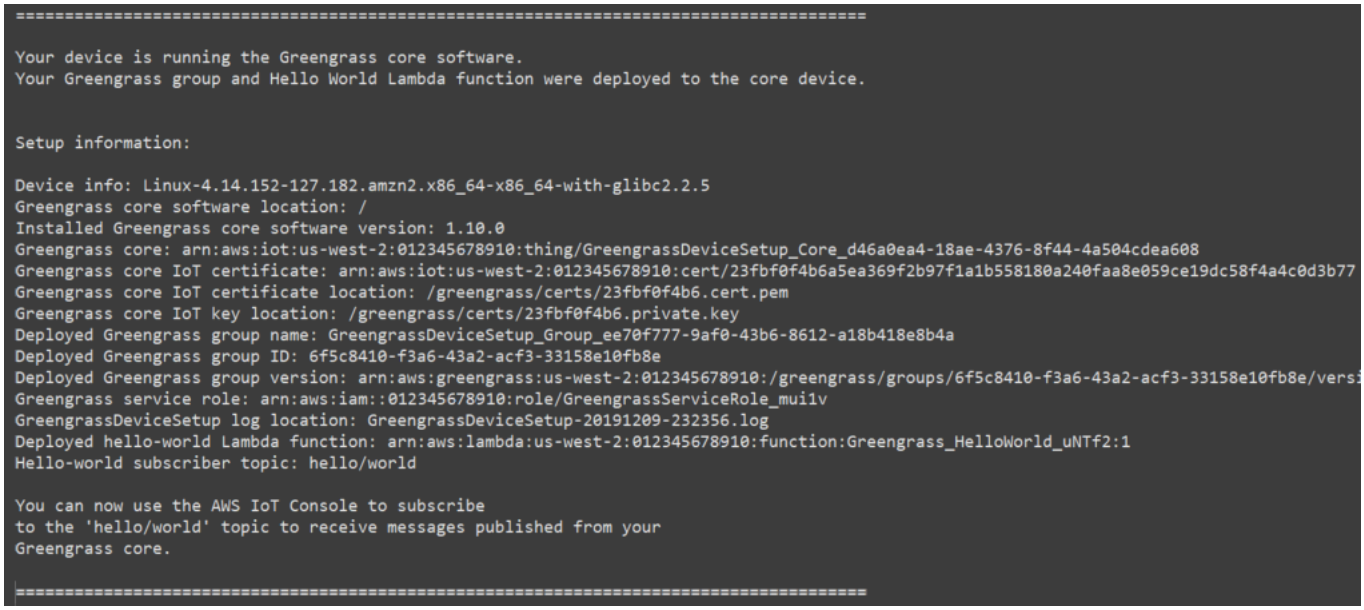
c. Reinicie el script.

   ```
   sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
   ```

d. Cuando se le pregunte si desea utilizar los valores de entrada de la sesión anterior o iniciar una nueva instalación, escriba `yes` para volver a utilizar los valores de entrada.

   > **Note**
   >
   > En las plataformas que soliciten que se reinicie, los valores de entrada de la sesión anterior, salvo las credenciales, se almacenan temporalmente en el archivo `GreengrassDeviceSetup.config.info`.

Una vez finalizada la configuración, el terminal muestra un mensaje de estado de operación correcta similar al siguiente.



4. Si ha incluido la función Hello World de Lambda, la configuración del dispositivo Greengrass implementa el grupo de Greengrass en el dispositivo principal. Para probar la función de Lambda u obtener información acerca de cómo eliminar la función de the section called “Verificación de la ejecución de la función de Lambda en el dispositivo central” del grupo, continúe hacia en el Módulo 3-1 del tutorial de introducción.

> **Note**
>
> Asegúrese de que la Región de AWS seleccionada en la consola sea la misma que utilizó para configurar el entorno de Greengrass. De forma predeterminada, la región es oeste de EE. UU. (Oregón).

Si no ha incluido la función Hello World de Lambda, puede crear su propia función de Lambda o probar otras características de Greengrass. Por ejemplo, puede añadir el conector de implementación de aplicaciones Docker al grupo y utilizarlo para implementar contenedores de Docker en el dispositivo principal.

Proporcionar las credenciales como valores de entrada

1. Descargue e inicie el script. Proporcione sus credenciales y cualquier otro valor de entrada que desee especificar. Los ejemplos siguientes muestran cómo proporcionar credenciales temporales, que incluyen el token de sesión.

   - Para utilizar todos los valores predeterminados:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   ```

   - Para especificar valores personalizados:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   --region us-east-1
   --group-name Custom_Group_Name
   --core-name Custom_Core_Name
   --ggc-root-path /custom/ggc/root/path
   --deployment-timeout 300
   --log-path /customized/log/path
   --hello-world-lambda
   --verbose
   ```

   > **Note**
   >
   > Si ejecuta la configuración del dispositivo Greengrass en una plataforma Raspbian u OpenWrt, copie sus credenciales. Debe proporcionarlos de nuevo después de reiniciar el dispositivo.

Para utilizar `curl` para descargar el script, reemplace `wget -q -O` con `curl` en el comando.

2. Si el dispositivo principal ejecuta Raspbian u OpenWrt, reinicie el dispositivo cuando se le solicite, proporcione las credenciales y, a continuación, reinicie el script.

   a. Cuando se le solicite que reinicie el dispositivo, ejecute uno de los siguientes comandos.

      Para plataformas Raspbian:

      ```
      sudo reboot
      ```

      Para plataformas OpenWrt:

      ```
      reboot
      ```

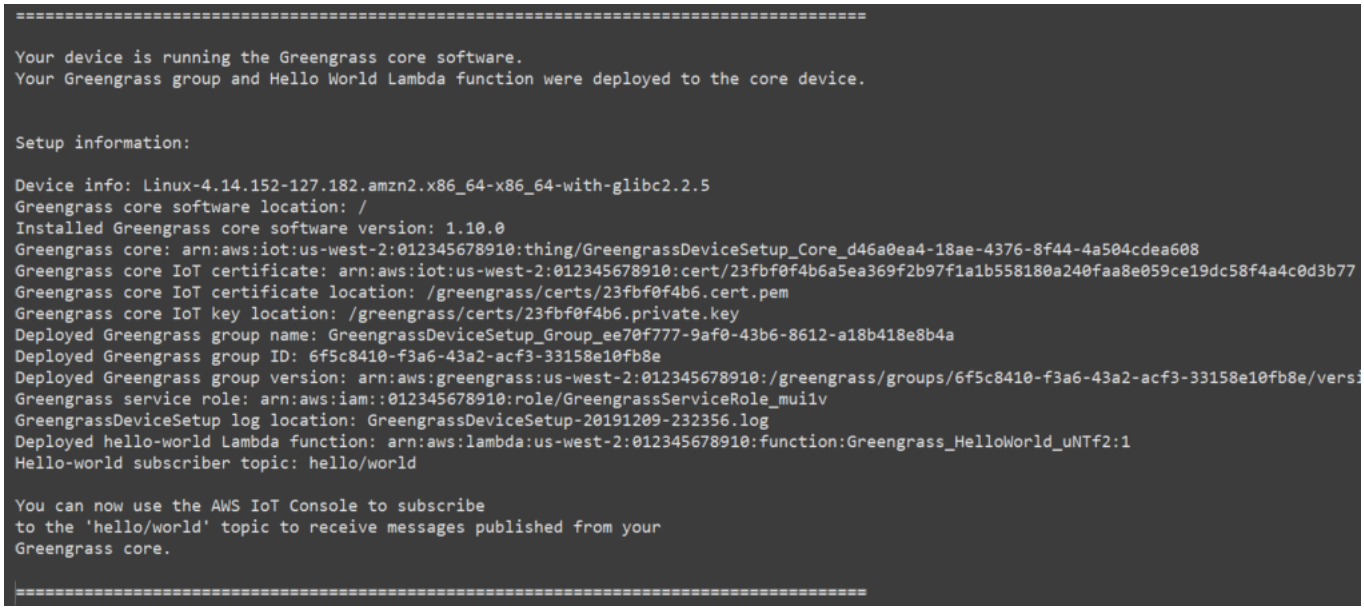
   b. Reinicie el script. Debe incluir sus credenciales en el comando, pero no en los demás valores de entrada. Por ejemplo:

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
      --aws-access-key-id AKIAIOSFODNN7EXAMPLE
      --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
      --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
      ```

   c. Cuando se le pregunte si desea utilizar los valores de entrada de la sesión anterior o iniciar una nueva instalación, escriba `yes` para volver a utilizar los valores de entrada.

      > **Note**
      >
      > En las plataformas que soliciten que se reinicie, los valores de entrada de la sesión anterior, salvo las credenciales, se almacenan temporalmente en el archivo `GreengrassDeviceSetup.config.info`.

Una vez finalizada la configuración, el terminal muestra un mensaje de estado de operación correcta similar al siguiente.



3. Si ha incluido la función Hello World de Lambda, la configuración del dispositivo Greengrass implementa el grupo de Greengrass en el dispositivo principal. Para probar la función de Lambda u obtener información acerca de cómo eliminar la función de the section called “Verificación de la ejecución de la función de Lambda en el dispositivo central” del grupo, continúe hacia en el Módulo 3-1 del tutorial de introducción.

> **Note**
>
> Asegúrese de que la Región de AWS seleccionada en la consola sea la misma que utilizó para configurar el entorno de Greengrass. De forma predeterminada, la región es oeste de EE. UU. (Oregón).

Si no ha incluido la función Hello World de Lambda, puede crear su propia función de Lambda o probar otras características de Greengrass. Por ejemplo, puede añadir el conector de implementación de aplicaciones Docker al grupo y utilizarlo para implementar contenedores de Docker en el dispositivo principal.

Módulo 1: Configuración del entorno para Greengrass

Este módulo le enseña a preparar un Raspberry Pi de serie, una instancia de Amazon EC2 o cualquier otro tipo de dispositivo para que AWS IoT Greengrass lo utilice como dispositivo de AWS IoT Greengrass de núcleo.

> **Tip**
>
> O bien, para utilizar un script que configure el dispositivo principal automáticamente, consulte the section called “Inicio rápido: configuración del dispositivo Greengrass”.

Completar este módulo debería tomarle menos de 30 minutos.

Antes de comenzar, lea los requisitos de este tutorial. A continuación, siga las instrucciones de configuración en uno de los siguientes temas. Elija solo el tema que se aplique al tipo de dispositivo principal.

Temas

- Configuración de un Raspberry Pi
- Configuración de una instancia de Amazon EC2
- Configuración de otros dispositivos

> **Note**
>
> Para obtener información sobre cómo utilizar AWS IoT Greengrass ejecutando en un contenedor de Docker preconfigurado, consulte the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”.

Configuración de un Raspberry Pi

Siga los pasos de este tema para configurar un Raspberry Pi para usarlo como un núcleo AWS IoT Greengrass.

> **Tip**
>
> AWS IoT Greengrass también proporciona otras opciones para instalar el software de AWS IoT Greengrass Core. Por ejemplo, puede utilizar la configuración de dispositivos Greengrass para configurar su entorno e instalar la versión más reciente del software AWS IoT Greengrass Core. O bien, en plataformas Debian compatibles, puede utilizar el administrador de paquetes APT para instalar o actualizar el software de AWS IoT Greengrass Core. Para obtener más información, consulte the section called “Instalación del software AWS IoT Greengrass Core”.

Si es la primera vez que configura un Raspberry Pi, debe seguir todos estos pasos. De lo contrario, puede ir directamente al paso 9. Sin embargo, le recomendamos que vuelva a instalar la imagen del sistema operativo de su Raspberry Pi, tal y como se recomienda en el paso 2.

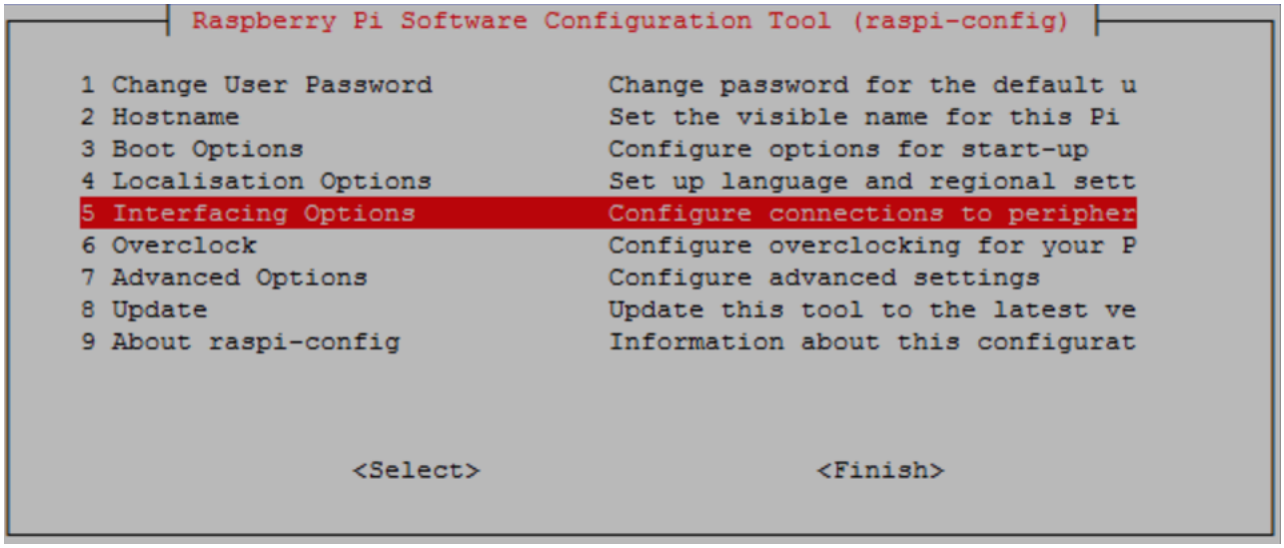
1. Descargue e instale una aplicación para formatear tarjetas SD, como SD Memory Card Formatter. Inserte la tarjeta SD en su equipo. Inicie el programa y elija la unidad donde insertó la tarjeta SD. Puede llevar a cabo un formateo rápido de la tarjeta SD.
2. Descargue el sistema operativo Raspbian Jessie como un archivo `zip`.
3. En una herramienta de escritura de tarjetas SD (como Etcher), siga las instrucciones para escribir el archivo `zip` descargado en la tarjeta SD. Dado que la imagen del sistema operativo es grande, este paso puede tardar un tiempo. Extraiga la tarjeta SD del equipo e inserte la tarjeta microSD en el Raspberry Pi.
4. Para la primera operación de inicio, le recomendamos que conecte el Raspberry Pi a un monitor (a través de HDMI), un teclado y un ratón. A continuación, conecte su Pi a una fuente de alimentación por microUSB para que se inicie el sistema operativo Raspbian.
5. Es posible que desee configurar la distribución del teclado del Pi antes de continuar. Para hacerlo, seleccione el icono de Raspberry en la esquina superior derecha, elija Preferences y, a continuación, elija Mouse and Keyboard Settings. A continuación, en la pestaña Keyboard, elija Keyboard Layout y luego elija una variante de teclado adecuada.
6. A continuación, conecte su Raspberry Pi a Internet a través de una red Wi-Fi o un cable Ethernet.

> **Note**
>
> Conecte su Raspberry Pi a la misma red que a la que está conectado su equipo y asegúrese de que tanto tu equipo como Raspberry Pi tienen acceso a Internet antes de continuar. Si está en un entorno de trabajo o detrás de un firewall, es posible que necesite conectar su dispositivo Pi y su equipo a la red de invitados para que ambos dispositivos estén en la misma red. Sin embargo, esta solución podría desconectar su equipo de los recursos de la red local, como la intranet. Otra solución consiste en conectar el dispositivo Pi y el equipo a la red wifi de invitados y conectar la red local a través de un cable Ethernet. Con esta configuración, el equipo debería poder conectarse al Raspberry Pi a través de la red wifi de invitados y a los recursos de la red local a través del cable Ethernet.

7. Debe configurar SSH en el Raspberry Pi para conectarse de forma remota. En su Raspberry Pi, abra una ventana de terminal y ejecute el siguiente comando:

```
sudo raspi-config
```

Debería ver lo siguiente:



Desplácese hacia abajo y elija Interfacing Options y, a continuación, elija P2 SSH. Cuando se le pregunte, elija Yes (Sí). (Utilice la tecla Tab seguida de Enter). Ahora, SSH debería estar habilitado. Seleccione OK (Aceptar). Utilice la tecla Tab para elegir Finish (Finalizar) y, a continuación, pulse Enter. Si Raspberry Pi no se reinicia automáticamente, ejecute el siguiente comando:

```
sudo reboot
```

8. En su Raspberry Pi ejecute el siguiente comando en el terminal:

```
hostname -I
```

Se devuelve la dirección IP de su Raspberry Pi.

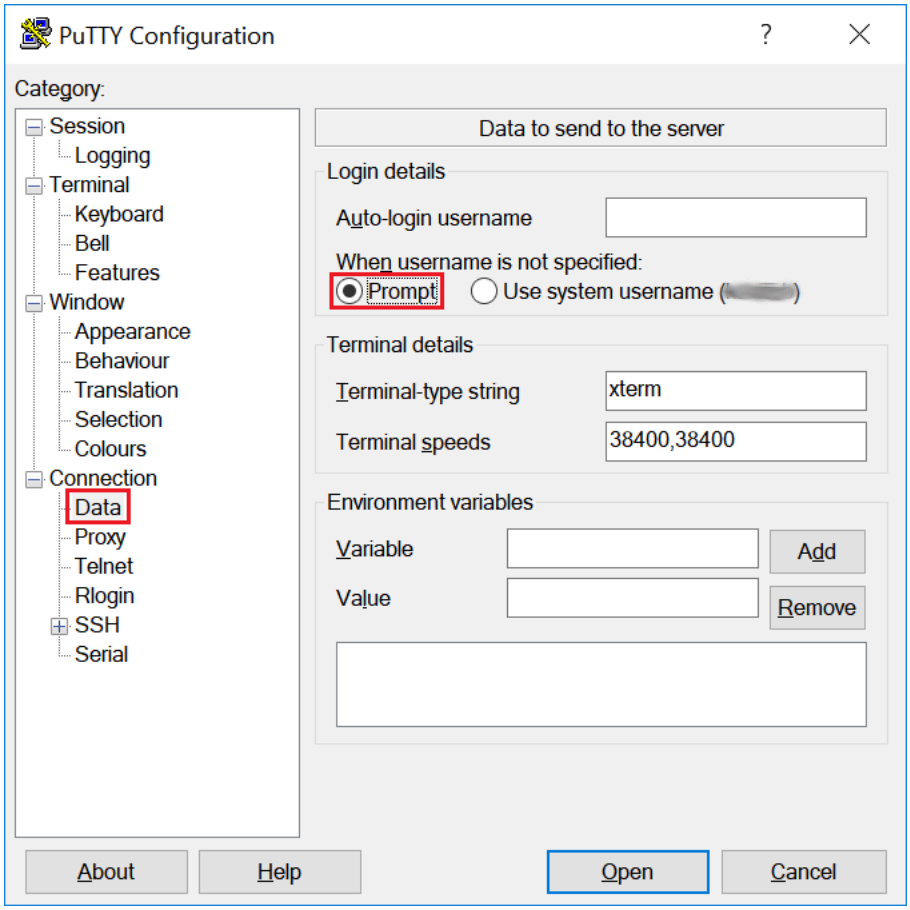
> **Note**
>
> Si, en el paso siguiente, recibe el mensaje (`Are you sure you want to continue connecting (yes/no)?`) relacionado con la huella digital de la clave ECDSA, escriba `yes`. La contraseña predeterminada del Raspberry Pi es **`raspberry`**.

Si utiliza macOS, abra una ventana de terminal y escriba lo siguiente:

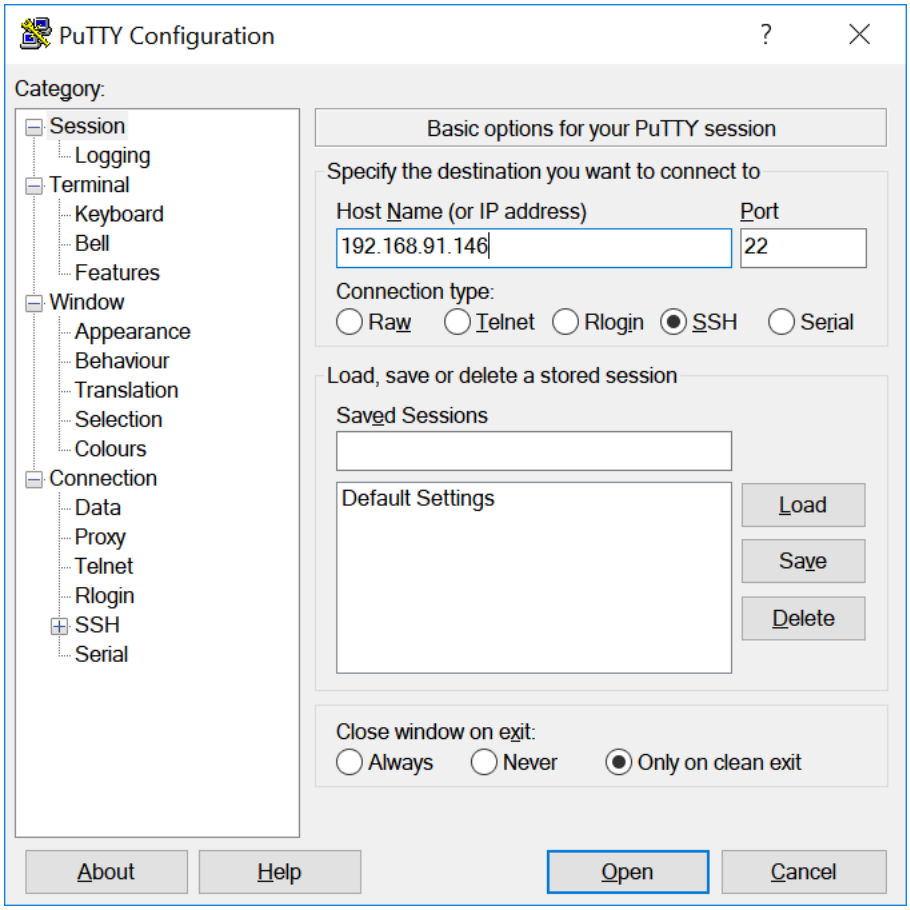
```
ssh pi@IP-address
```

`Dirección-IP` es la dirección IP de su Raspberry Pi que obtuvo utilizando el comando `hostname -I`.

Si utiliza Windows, tendrá que instalar y configurar PuTTY. Amplíe Connection, elija Data y asegúrese de que Prompt está seleccionado:

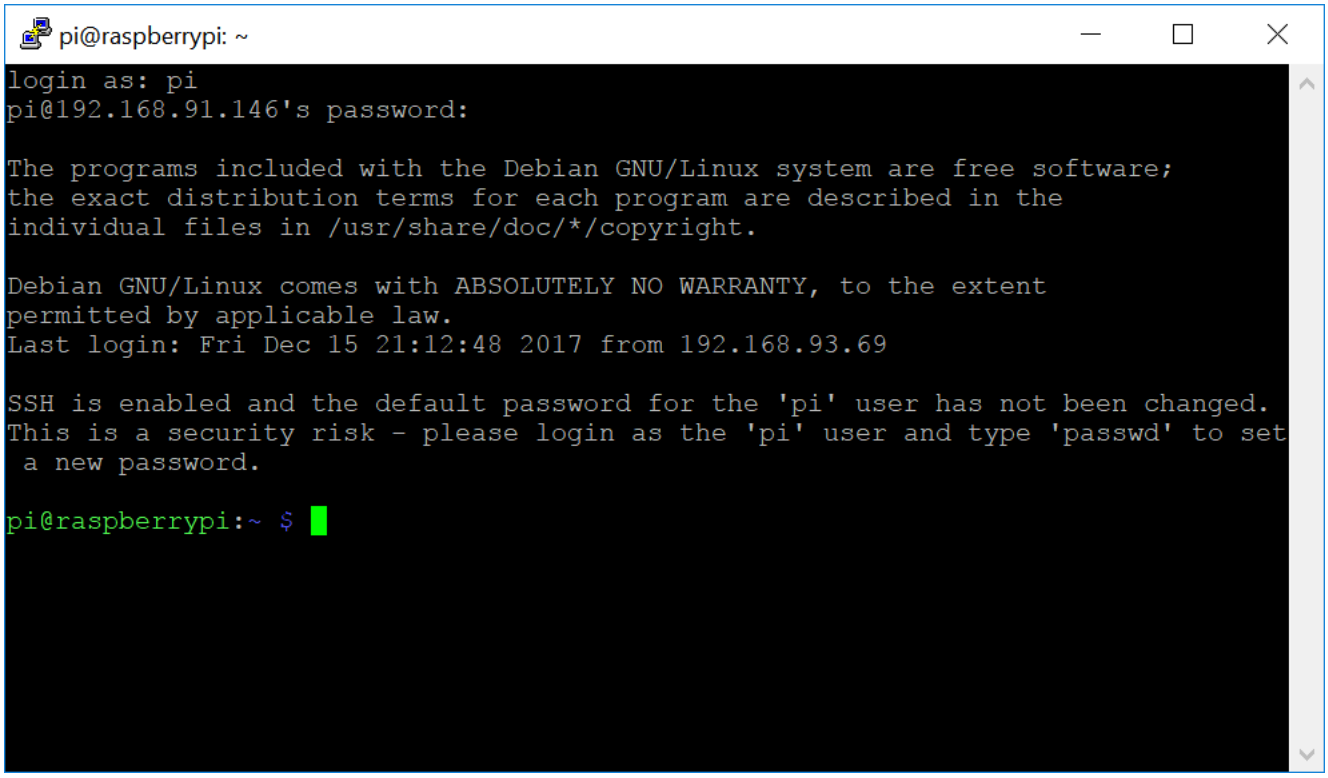


A continuación, elija Session, escriba la dirección IP del Raspberry Pi y, a continuación, elija Open con la configuración predeterminada.



Si se muestra una alerta de seguridad de PuTTY, seleccione Yes.

El nombre de inicio de sesión y la contraseña predeterminados del Raspberry Pi son **`pi`** y **`raspberry`**, respectivamente.



> **Note**
>
> Si el equipo está conectado a una red remota mediante VPN, podría haber problemas para conectarse desde el equipo al Raspberry Pi mediante SSH.

9. Ahora está preparado para configurar el Raspberry Pi para AWS IoT Greengrass. En primer lugar, ejecute los siguientes comandos desde una ventana de terminal de Raspberry Pi local o una ventana de terminal de SSH:

> **Tip**
>
> AWS IoT Greengrass también proporciona otras opciones para instalar el software de AWS IoT Greengrass Core. Por ejemplo, puede utilizar la configuración de dispositivos Greengrass para configurar su entorno e instalar la versión más reciente del software AWS IoT Greengrass Core. O bien, en plataformas Debian compatibles, puede utilizar el administrador de paquetes APT para instalar o actualizar el software de AWS IoT Greengrass Core. Para obtener más información, consulte the section called “Instalación del software AWS IoT Greengrass Core”.

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

10. Para mejorar la seguridad en el dispositivo Raspberry Pi, habilite la protección de enlaces permanentes y simbólicos (symlink) al inicio del sistema operativo.

    a. Vaya al archivo `98-rpi.conf`.

    ```
    cd /etc/sysctl.d
    ls
    ```

    > **Note**
    >
    > Si no ve el archivo `98-rpi.conf`, siga las instrucciones del archivo `README.sysctl`.

    b. Utilice un editor de texto (como, por ejemplo, Leafpad, GNU nano o vi) para añadir las dos líneas siguientes al final del archivo. Es posible que tenga que utilizar el comando `sudo` para editar como raíz (por ejemplo, `sudo nano 98-rpi.conf`).

    ```
    fs.protected_hardlinks = 1
    fs.protected_symlinks = 1
    ```

    c. Reinicie el dispositivo Pi.

    ```
    sudo reboot
    ```

    Pasado un minuto aproximadamente, conecte el Pi utilizando SSH y, a continuación, ejecute el comando siguiente para confirmar el cambio:

    ```
    sudo sysctl -a 2> /dev/null | grep fs.protected
    ```

    Debería ver `fs.protected_hardlinks = 1` y `fs.protected_symlinks = 1`.

11. Edite el archivo de arranque de la línea de comandos para habilitar y montar los grupos de control de memoria. De este modo, AWS IoT Greengrass puede establecer el límite de memoria

de las funciones Lambda. Los grupos C también deben ejecutar AWS IoT Greengrass en el modo de creación de contenedores predeterminado.

a. Vaya a su directorio `boot`.

```
cd /boot/
```

b. Utilice un editor de texto para abrir `cmdline.txt`. Añada lo siguiente al final de la línea existente, no como nueva línea. Es posible que tenga que utilizar el comando `sudo` para editar como raíz (por ejemplo, `sudo nano cmdline.txt`).

```
cgroup_enable=memory cgroup_memory=1
```

c. Ahora reinicie el dispositivo Pi.

```
sudo reboot
```

Su Raspberry Pi ya debería estar preparado para AWS IoT Greengrass.

12. Opcional. Instale Java 8 Runtime, requerido por el administrador de secuencias. En este tutorial no se utiliza el administrador de secuencias, pero sí el flujo de trabajo Creación predeterminada de un grupo, que habilita el administrador de secuencias de forma predeterminada. Utilice los siguientes comandos para instalar Java 8 Runtime en el dispositivo principal o desactivar el administrador de secuencias antes de implementar el grupo. Las instrucciones para desactivar el administrador de secuencias se proporcionan en el Módulo 3.

```
sudo apt install openjdk-8-jdk
```

13. Para asegurarse de que dispone de todas las dependencias necesarias, descargue y ejecute el comprobador de dependencias Greengrass desde el repositorio de AWS IoT Greengrass Muestras de GitHub. Estos comandos descomprimen y ejecutan el script comprobador de dependencias en el directorio `Downloads`.

> **Note**
>
> El comprobador de dependencias puede fallar si está ejecutando la versión 5.4.51 del núcleo de Raspbian. Esta versión no monta los grupos de control de memoria correctamente. Esto podría provocar un error en las funciones de Lambda que se ejecutan en modo contenedor.

Para obtener más información sobre cómo actualizar el núcleo, consulte los Cgroups que no se cargan tras la actualización del núcleo en los foros de Raspberry Pi.

```
cd /home/pi/Downloads
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

Donde aparece `more`, pulse la tecla Spacebar para mostrar otra pantalla de texto.

Important

En necesario que el tiempo de ejecución de Python 3.7 ejecute las funciones de Lambda locales para realizar este tutorial. Cuando el administrador de secuencias esté habilitado, también es necesario Java 8 Runtime. Si el script `check_ggc_dependencies` genera advertencias acerca de la ausencia de estos requisitos previos relativos al tiempo de ejecución, asegúrese de instalarlos antes de continuar. Puede hacer caso omiso de las advertencias que indican que faltan los requisitos previos relativos a tiempos de ejecución opcionales.

Para obtener más información acerca del comando modprobe, ejecute man modprobe en el terminal.

Ya está terminada la configuración del dispositivo Raspberry Pi. Siga en the section called “Módulo 2: Instalación del software de AWS IoT Greengrass Core”.

Configuración de una instancia de Amazon EC2

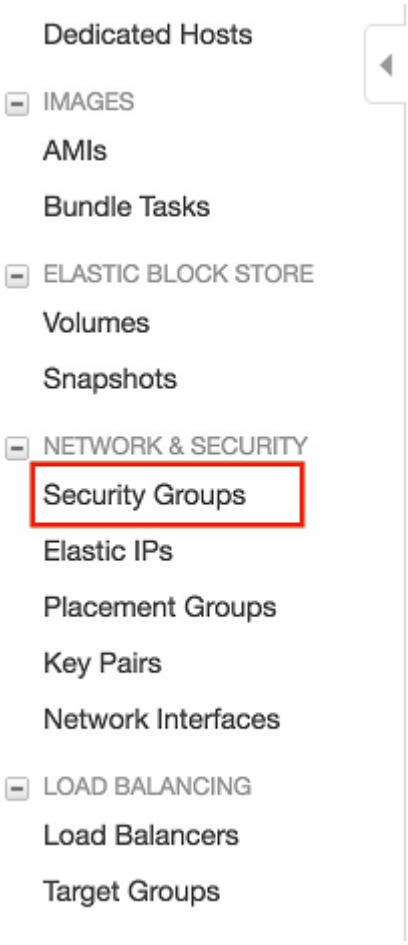
Siga los pasos de este tema para configurar una instancia de Amazon EC2 para utilizarla como su núcleo de AWS IoT Greengrass .

> **Tip**
>
> O bien, para usar un script que configure su entorno e instale el software AWS IoT Greengrass principal por usted, consultethe section called “Inicio rápido: configuración del dispositivo Greengrass”.

Aunque puede completar este tutorial con una instancia de Amazon EC2, lo ideal es AWS IoT Greengrass que se utilice con hardware físico. Le recomendamos que configure un Raspberry Pi en lugar de utilizar una instancia de Amazon EC2 cuando sea posible. Si utiliza un dispositivo Raspberry Pi, no tiene que seguir los pasos de este tema.

1. Inicie sesión en la AWS Management Console y lance una instancia de Amazon EC2 mediante una AMI de Amazon Linux. Para obtener más información acerca de las instancias de Amazon EC2, consulte la Guía de introducción de Amazon EC2.
2. Cuando la instancia de Amazon EC2 esté en ejecución, habilite el puerto 8883 para permitir las comunicaciones MQTT entrantes de forma que otros dispositivos puedan conectarse al núcleo. AWS IoT Greengrass

   a. En el panel de navegación de la consola de Amazon EC2, seleccione Grupos de seguridad.



b. Seleccione el grupo de seguridad para la instancia que acaba de lanzar y, a continuación, elija la pestaña Reglas de entrada.

c. Elija Editar reglas de entrada.

   Para habilitar el puerto 8883, añada una regla TCP personalizada al grupo de seguridad. Para obtener más información, consulte Añadir reglas a un grupo de seguridad en la Guía del usuario de Amazon EC2.

d. En la página Editar reglas de entrada, seleccione Añadir regla, introduzca la siguiente configuración y, a continuación, seleccione Guardar.

   - En Tipo, elija Regla TCP personalizada.
   - En Rango de puertos, escriba **`8883`**.
   - En Fuente, elija Cualquiera.
   - En Descripción, escriba **`MQTT Communications`**.

3. Conéctese a la instancia de Amazon EC2.

   a. En el panel de navegación, elija Instances (Instancias), seleccione la suya y, después, elija Connect (Conectarse).

   b. Siga las instrucciones de la página Connect To Your Instance (Conectar a su instancia) para conectar a su instancia utilizando SSH y su archivo de clave privada.

   Puede utilizar PuTTY para Windows o Terminal para macOS. Para obtener más información, consulte Conectarse a su instancia de Linux en la Guía del usuario de Amazon EC2.

   Ya está listo para configurar su instancia de Amazon EC2 para AWS IoT Greengrass.

4. Una vez conectado a su instancia de Amazon EC2, cree las cuentas `ggc_user` y `ggc_group`:

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

> **Note**
>
> Si el comando `adduser` no está disponible en su sistema, utilice el siguiente comando.
>
> ```
> sudo useradd --system ggc_user
> ```

5. Para mejorar la seguridad, asegúrese de que estén habilitadas las protecciones de enlaces permanentes y simbólicos (symlink) al inicio del sistema operativo de la instancia de Amazon EC2 en startup.

> **Note**
>
> Los pasos para habilitar la protección de enlaces permanentes y simbólicos varían según el sistema operativo. Consulte la documentación de su distribución.

   a. Ejecute el siguiente comando para comprobar si están habilitadas las protecciones de enlaces permanentes y simbólicos:

```
sudo sysctl -a | grep fs.protected
```

Si los enlaces permanentes y los enlaces simbólicos están establecidos en 1, las protecciones están habilitadas correctamente. Continúe con el paso 6.

> **Note**
>
> Los enlaces simbólicos están representados por `fs.protected_symlinks`.

b. Si los enlaces permanentes y los enlaces simbólicos no están configurados en 1, habilite estas protecciones. Vaya al archivo de configuración del sistema.

```
cd /etc/sysctl.d
ls
```

c. Utilice su editor de texto preferido (por ejemplo, Leafpad, GNU nano o vi) para agregar las dos líneas siguientes al final del archivo de configuración del sistema. En Amazon Linux 1, es el archivo `00-defaults.conf`. En Amazon Linux 2, es el archivo `99-amazon.conf`. Es posible que tenga que cambiar los permisos del archivo (con el comando `chmod`) para poder escribir en él, o bien usar el comando `sudo` para editarlo como raíz (por ejemplo, `sudo nano 00-defaults.conf`).

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

d. Reinicie la instancia de Amazon EC2.

```
sudo reboot
```

Pasados unos minutos, conecte su instancia utilizando SSH y, a continuación, ejecute el comando siguiente para confirmar el cambio.

```
sudo sysctl -a | grep fs.protected
```

Debería ver los enlaces permanentes y simbólicos establecidos en 1.

6. Extraiga y ejecute el siguiente script para montar grupos de control de Linux (cgroups). Esto permite AWS IoT Greengrass establecer el límite de memoria para las funciones Lambda. Los grupos C también deben ejecutarse AWS IoT Greengrass en el modo de contenerización predeterminado.

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-
mount/951c38ee8d802330454bdede20d85ec1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

Su instancia de Amazon EC2 ahora debería estar preparada para AWS IoT Greengrass.

7. Opcional. Instale Java 8 Runtime, requerido por el administrador de secuencias. En este tutorial no se utiliza el administrador de secuencias, pero sí el flujo de trabajo Creación predeterminada de un grupo, que habilita el administrador de secuencias de forma predeterminada. Utilice los siguientes comandos para instalar Java 8 Runtime en el dispositivo principal o desactivar el administrador de secuencias antes de implementar el grupo. Las instrucciones para desactivar el administrador de secuencias se proporcionan en el Módulo 3.

   - Para distribuciones basadas en Debian:

     ```
     sudo apt install openjdk-8-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-1.8.0-openjdk
     ```

8. Para asegurarse de que dispone de todas las dependencias necesarias, descargue y ejecute el comprobador de dependencias de Greengrass desde AWS IoT Greengrass el repositorio de muestras en adelante. GitHub Estos comandos descargan, descomprimen y ejecutan el script comprobador de dependencias en la instancia de Amazon EC2.

   ```
   mkdir greengrass-dependency-checker-GGCv1.11.x
   cd greengrass-dependency-checker-GGCv1.11.x
   wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
   dependency-checker-GGCv1.11.x.zip
   unzip greengrass-dependency-checker-GGCv1.11.x.zip
   cd greengrass-dependency-checker-GGCv1.11.x
   sudo ./check_ggc_dependencies | more
   ```

   Important

   En necesario que el tiempo de ejecución de Python 3.7 ejecute las funciones de Lambda locales para realizar este tutorial. Cuando el administrador de secuencias esté habilitado,

también es necesario Java 8 Runtime. Si el script `check_ggc_dependencies` genera advertencias acerca de la ausencia de estos requisitos previos relativos al tiempo de ejecución, asegúrese de instalarlos antes de continuar. Puede hacer caso omiso de las advertencias que indican que faltan los requisitos previos relativos a tiempos de ejecución opcionales.

Ya está terminada la configuración de la instancia de Amazon EC2. Siga en the section called "Módulo 2: Instalación del software de AWS IoT Greengrass Core".

Configuración de otros dispositivos

Siga los pasos de este tema para configurar un dispositivo (distinto de un Raspberry Pi) para utilizarlo como su núcleo de AWS IoT Greengrass.

> **Tip**
>
> O bien, para utilizar un script que configure el entorno e instale el software AWS IoT Greengrass Core automáticamente, consulte the section called "Inicio rápido: configuración del dispositivo Greengrass".

Si es la primera vez que usa AWS IoT Greengrass, le recomendamos que use un Raspberry Pi o una instancia de Amazon EC2 como dispositivo del núcleo y que siga los pasos de configuración apropiados para el dispositivo.

Si planea construir un sistema personalizado basado en Linux utilizando el Proyecto Yocto, puede usar la receta AWS IoT Greengrass Bitbake del proyecto `meta-aws`. Esta receta también le ayuda a desarrollar una plataforma de software compatible con el software de AWS periférico para aplicaciones integradas. La versión de Bitbake instala, configura y ejecuta automáticamente el software de AWS IoT Greengrass Core en tu dispositivo.

Tecnología del proyecto Yocto

Un proyecto de colaboración de código abierto que le ayuda a crear sistemas personalizados basados en Linux para aplicaciones integradas, independientemente de la arquitectura de hardware. Para obtener más información, consulte el Proyecto Yocto.

`meta-aws`

Un proyecto AWS gestionado que proporciona recetas de Yocto. Puede utilizar las recetas para desarrollar software periférico de AWS avanzado en sistemas basados en Linux creados con OpenEmbedded y Proyecto Yocto. Para obtener más información sobre esta capacidad compatible con la comunidad, consulte el proyecto `meta-aws` en GitHub.

`meta-aws-demos`

Un proyecto AWS gestionado que contiene demostraciones del proyecto `meta-aws`. Para ver más ejemplos sobre el proceso de integración, consulta el proyecto `meta-aws-demos` en GitHub.

Si desea usar otro dispositivo o plataforma compatible, siga los pasos de este tema.

1. Si tiene un dispositivo NVIDIA Jetson, primero debe instalar el firmware con el instalador de JetPack 4.3. Si está configurando un dispositivo diferente, vaya al paso 2.

   > **Note**
   >
   > La versión del instalador JetPack que se utiliza depende de la versión del conjunto de herramientas CUDA de destino. En las instrucciones siguientes se se usa JetPack 4.3 y CUDA Toolkit 10.0. Para obtener información sobre el uso de las versiones adecuadas para el dispositivo, consulte How to Install Jetpack en la documentación de NVIDIA.

   a. En los equipos físicos de escritorio que ejecutan Ubuntu 16.04 o versiones posteriores, actualice el firmware con el instalador de JetPack 4.3, tal y como se describe en el artículo Download and Install JetPack (4.3) de la documentación de NVIDIA.

      Siga las instrucciones del instalador para instalar todos los paquetes y las dependencias en la placa Jetson, que debe estar conectada al escritorio con un cable Micro-B.

   b. Reinicie la placa en modo normal y conecte una pantalla.

      > **Note**
      >
      > Cuando utilice SSH para conectarse a la placa Jetson, utilice el nombre de usuario predeterminado (**`nvidia`**) y la contraseña predeterminada (**`nvidia`**).

2. Ejecute los siguientes comandos para crear el usuario `ggc_user` y el grupo `ggc_group`. Los comandos que ejecuta difieren, en función de la distribución instalada en su dispositivo del núcleo.

   - Si el dispositivo del núcleo está ejecutando OpenWrt, ejecute los siguientes comandos:

     ```
     opkg install shadow-useradd
     opkg install shadow-groupadd
     useradd --system ggc_user
     groupadd --system ggc_group
     ```

   - De lo contrario, ejecute los comandos siguientes:

     ```
     sudo adduser --system ggc_user
     sudo addgroup --system ggc_group
     ```

     > **Note**
     >
     > Si el comando `addgroup` no está disponible en su sistema, utilice el siguiente comando.
     >
     > ```
     > sudo groupadd --system ggc_group
     > ```

3. Opcional. Instale Java 8 Runtime, requerido por el administrador de secuencias. En este tutorial no se utiliza el administrador de secuencias, pero sí el flujo de trabajo Creación predeterminada de un grupo, que habilita el administrador de secuencias de forma predeterminada. Utilice los siguientes comandos para instalar Java 8 Runtime en el dispositivo principal o desactivar el administrador de secuencias antes de implementar el grupo. Las instrucciones para desactivar el administrador de secuencias se proporcionan en el Módulo 3.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install openjdk-8-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-1.8.0-openjdk
     ```

4. Para asegurarse de que dispone de todas las dependencias necesarias, descargue y ejecute el comprobador de dependencias Greengrass desde el repositorio de AWS IoT Greengrass Muestras de GitHub. Estos comandos descomprimen y ejecutan el script comprobador de dependencias.

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

> **Note**
>
> El script de `check_ggc_dependencies` se ejecuta en plataformas compatibles con AWS IoT Greengrass y requiere comandos específicos del sistema Linux. Para obtener más información, consulte el archivo Readme del comprobador de dependencias.

5. Instale en su dispositivo todas las dependencias necesarias, tal y como indica el resultado del comprobador de dependencias. En caso de que falten dependencias en el nivel del kernel, es posible que tenga que volver a compilar el kernel. Para montar grupos de control de Linux (`cgroups`), puede ejecutar el script cgroupfs-mount. De este modo, AWS IoT Greengrass puede establecer el límite de memoria de las funciones Lambda. Los grupos C también deben ejecutar AWS IoT Greengrass en el modo de creación de contenedores predeterminado.

   Si no hay errores en la salida, AWS IoT Greengrass debería poder ejecutarse correctamente en el dispositivo.

> **Important**
>
> En necesario que el tiempo de ejecución de Python 3.7 ejecute las funciones de Lambda locales para realizar este tutorial. Cuando el administrador de secuencias esté habilitado, también es necesario Java 8 Runtime. Si el script `check_ggc_dependencies` genera advertencias acerca de la ausencia de estos requisitos previos relativos al tiempo de ejecución, asegúrese de instalarlos antes de continuar. Puede hacer caso omiso de las advertencias que indican que faltan los requisitos previos relativos a tiempos de ejecución opcionales.

Para ver la lista de requisitos y dependencias de AWS IoT Greengrass, consulte the section called “Plataformas compatibles y requisitos”.

Módulo 2: Instalación del software de AWS IoT Greengrass Core

Este módulo le enseña a instalar el software de AWS IoT Greengrass Core en el dispositivo que desee. En él, primero debe crear un grupo y un núcleo de Greengrass. A continuación, debe descargar, configurar e iniciar el software en el dispositivo principal. Para obtener más información acerca de la funcionalidad del software de AWS IoT Greengrass Core, consulte the section called “Configuración de AWS IoT Greengrass Core”.

Antes de comenzar, asegúrese de haber completado los pasos de configuración del Módulo 1 para el dispositivo elegido.

> **Tip**
>
> AWS IoT Greengrass también proporciona otras opciones para instalar el software de AWS IoT Greengrass Core. Por ejemplo, puede utilizar la configuración de dispositivos Greengrass para configurar su entorno e instalar la versión más reciente del software AWS IoT Greengrass Core. O bien, en plataformas Debian compatibles, puede utilizar el administrador de paquetes APT para instalar o actualizar el software de AWS IoT Greengrass Core. Para obtener más información, consulte the section called “Instalación del software AWS IoT Greengrass Core”.

Completar este módulo debería tomarle menos de 30 minutos.

Temas

- Aprovisione un objeto AWS IoT para usarlo como núcleo de Greengrass
- Crea un grupo AWS IoT Greengrass para el núcleo
- Instale y ejecute AWS IoT Greengrass en el dispositivo central

Aprovisione un objeto AWS IoT para usarlo como núcleo de Greengrass

Los núcleos de Greengrass son dispositivos que ejecutan el software AWS IoT Greengrass Core para gestionar los procesos locales de IoT. Para configurar un núcleo de Greengrass, se crea un objeto AWS IoT que represente un dispositivo o entidad lógica que se conecte a AWS IoT. Al registrar un dispositivo como un objeto AWS IoT, ese dispositivo puede usar un certificado digital y claves que le permiten acceder a AWS IoT. Utiliza una política de AWS IoT para permitir que el dispositivo se comunique con los servicios de AWS IoT y AWS IoT Greengrass.

En esta sección, registra su dispositivo como un objeto AWS IoT para usarlo como núcleo de Greengrass.

Crear un objeto de AWS IoT

1. Vaya a la consola de AWS IoT.
2. En Administrar, expanda Todos los dispositivos y, a continuación, elija Objetos.
3. En la página Objetos, seleccione Crear ojetos.
4. En la página Crea objetos, elija Crear un solo objeto, y luego seleccione Siguiente.
5. En la página Especificar propiedades del objeto, haga lo siguiente:

   a. En Nombre del objeto, introduzca un nombre que represente su dispositivo, como **`MyGreengrassV1Core`**.

   b. Elija Next (Siguiente).
6. En la página Configurar el certificado del dispositivo, seleccione Siguiente.
7. En la página Adjuntar políticas al certificado, realice uno de los siguientes procedimientos:
   - Seleccione una política existente que conceda los permisos que requieren los dispositivos cliente y, a continuación, seleccione Crear objeto.

     Se abre un modal en el que puede descargar los certificados y las claves que el dispositivo utiliza para conectarse a la Nube de AWS.
   - Cree y adjunte una nueva política que conceda permisos al dispositivo principal. Haga lo siguiente:

     a. Elija Create Policy (Crear política).

        La página Create policy (Crear política) se abre en una pestaña nueva.

     b. En la página Create policy (Crear política), haga lo siguiente:

i. En Nombre de la política, introduzca un nombre que describa la política, como **`GreengrassV1CorePolicy`**.

ii. En la pestaña Declaraciones de política, en Documento de política, seleccione JSON.

iii. Ingrese el siguiente documento de política. Esta política permite que el núcleo se comunique con el servicio AWS IoT Core, interactúe con las sombras de dispositivos y se comunique con el servicio AWS IoT Greengrass. Para obtener información sobre cómo restringir el acceso a esta política en función de su caso de uso, consulte Política mínima de AWS IoT para el dispositivo central de AWS IoT Greengrass.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. Elija Create (Crear) para crear la política.

c. Vuelva a la pestaña del navegador con la página Adjuntar políticas al certificado abierta. Haga lo siguiente:

i. En la lista de Políticas, seleccione la política que ha creado, como por ejemplo GreengrassV1CorePolicy.

Si no se puede ver la política, seleccione el botón de actualizar.

ii. Elija Crear objeto.

Se abre un modal en el que puede descargar los certificados y las claves que el núcleo utiliza para conectarse a AWS IoT.

8. Vuelva a la pestaña del navegador con la página Adjuntar políticas al certificado abierta. Haga lo siguiente:

a. En la lista de Políticas, seleccione la política que ha creado, como por ejemplo GreengrassV1CorePolicy.

Si no se puede ver la política, seleccione el botón de actualizar.

b. Elija Crear objeto.

Se abre un modal en el que puede descargar los certificados y las claves que el núcleo utiliza para conectarse a AWS IoT.

9. En el modal Descargar certificados y claves, descargue los certificados del dispositivo.

> **Important**
>
> Descargue los recursos de seguridad antes de elegir Listo.

Haga lo siguiente:

a. Para el certificado del dispositivo, seleccione Descargar para descargar el certificado del dispositivo.

b. En Archivo de clave pública, seleccione Descargar para descargar la clave pública del certificado.

c. En Archivo de clave privada, seleccione Descargar para descargar el archivo de clave privada del certificado.

d. Revise la Autenticación de servidor en la Guía del desarrollador de AWS IoT y seleccione el certificado de CA raíz adecuado. Le recomendamos que utilice los puntos de conexión de Amazon Trust Services (ATS) y los certificados de CA raíz de ATS. En Certificados de CA raíz, seleccione Descargar para obtener un certificado de CA raíz.

e. Seleccione Done (Listo).

Tome nota del identificador del certificado que comparten los nombres de archivo del certificado y las claves del dispositivo. Lo necesitará más adelante.

Crea un grupo AWS IoT Greengrass para el núcleo

los grupos AWS IoT Greengrass contienen la configuración y otra información sobre sus componentes, como los dispositivos cliente, las funciones de Lambda y los conectores. Un grupo define la configuración de un núcleo, incluida la forma en que sus componentes pueden interactuar entre sí.

En esta sección creará un grupo para el núcleo.

> **Tip**
>
> Para ver un ejemplo que utiliza la API AWS IoT Greengrass para crear e implementar un grupo, consulte el repositorio gg_group_setup de GitHub.

Crea un grupo para el núcleo

1. Vaya a la consola de AWS IoT.

2. En Administrar, expanda los dispositivos Greengrass y elija Grupos (V1).

> **Note**
>
> Si no ve el menú de dispositivos Greengrass, cambie a un Región de AWS que admita AWS IoT Greengrass V1. Para ver una lista completa de las regiones admitidas, consulte AWS IoT Greengrass V1Puntos de conexión y cuotas en la Referencia general de AWS. Debe crear el objetoAWS IoT para su núcleo en una región en la que AWS IoT Greengrass V1 esté disponible.

3. En la página de grupos de Greengrass, seleccione Crear grupo.
4. En la página Crear grupo de Greengrass, haga lo siguiente:

   a. Para el nombre del grupo Greengrass, introduzca un nombre que describa el grupo, como por ejemplo **`MyGreengrassGroup`**.

   b. Para el núcleo de Greengrass, elijaAWS IoT lo que creó anteriormente, como MyGreengrassV1Core.

      La consola selecciona automáticamente el certificado de dispositivo del dispositivo.

   c. Elija Create group.

Instale y ejecute AWS IoT Greengrass en el dispositivo central

> **Note**
>
> En este tutorial, se proporcionan instrucciones para ejecutar el software de AWS IoT Greengrass Core en un Raspberry Pi, pero puede utilizar cualquier dispositivo compatible.

En esta sección, configurará, instalará y ejecutará el software de AWS IoT Greengrass Core en su dispositivo principal.

Cómo instalar y ejecutar AWS IoT Greengrass

1. En la sección Software de AWS IoT Greengrass Core de esta guía, descargue el paquete de instalación del software AWS IoT Greengrass Core. Elija el paquete que mejor se adapte a la arquitectura de la CPU, la distribución y el sistema operativo del dispositivo principal.

- Para Raspberry Pi, descargue el paquete para la arquitectura de ARMv7L y el sistema operativo Linux.
- Para una instancia Amazon EC2, descargue el paquete para la arquitectura x86_64 y el sistema operativo Linux.
- En el caso de NVIDIA Jetson TX2, descargue el paquete para la arquitectura Armv8 (AArch64) y el sistema operativo Linux.
- En el caso de Intel Atom, descargue el paquete para la arquitectura x86_64 y el sistema operativo Linux.

2. En pasos anteriores, descargó cinco archivos en su equipo:

- `greengrass-`*`OS-architecture`*`-1.11.6.tar.gz`: Este archivo comprimido contiene el software de AWS IoT Greengrass Core que se ejecuta en el dispositivo del núcleo.
- *`certificateId`*`-certificate.pem.crt` – El certificado de dispositivo.
- *`certificateId`*`-public.pem.key` – El archivo de clave pública del certificado del dispositivo.
- *`certificateId`*`-private.pem.key` – El archivo de clave privada del certificado del dispositivo.
- `AmazonRootCA1.pem` – El archivo de la entidad de certificación (CA) raíz de Amazon.

En este paso, transfiere estos archivos del equipo al dispositivo del núcleo. Haga lo siguiente:

a. Si no conoce la dirección IP de su dispositivo de Greengrass Core, abra un terminal en el dispositivo central y ejecute el siguiente comando:

> **Note**
>
> Este comando podría no devolver la dirección IP correcta de algunos dispositivos. Consulte la documentación de su dispositivo para recuperar la dirección IP del mismo.

```
hostname -I
```

b. Transfiera estos archivos desde el ordenador al dispositivo del núcleo. Los pasos de File Transfer varían según el sistema operativo de su equipo. Elija su sistema operativo para ver los pasos que muestran cómo transferir archivos a su dispositivo Raspberry Pi.

> **Note**
>
> En Raspberry Pi, el nombre de usuario y la contraseña predeterminados son **`pi`** y **`raspberry`**, respectivamente.
> En NVIDIA Jetson TX2, el nombre de usuario y la contraseña predeterminados son **`nvidia`** y **`nvidia`**, respectivamente.

Windows

Para transferir los archivos comprimidos desde el equipo a un dispositivo principal de Raspberry Pi, utilice una herramienta adecuada, como WinSCP o el comando pscp de PuTTY. Para utilizar el comando pscp, abra una ventana del símbolo de sistema en el equipo y ejecute lo siguiente:

```
cd path-to-downloaded-files
pscp -pw Pi-password greengrass-OS-architecture-1.11.6.tar.gz pi@IP-
address:/home/pi
pscp -pw Pi-password certificateId-certificate.pem.crt pi@IP-address:/home/
pi
pscp -pw Pi-password certificateId-public.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-private.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> El número de versión de este comando debe coincidir con la versión del paquete de software AWS IoT Greengrass Core.

macOS

Para transferir los archivos comprimidos desde un Mac a un dispositivo Raspberry Pi central, abra una ventana de terminal en el equipo y ejecute los comandos siguientes. La *ruta-hacia-archivos-descargados* suele ser `~/Downloads`.

> **Note**
>
> Es posible que se le soliciten dos contraseñas. En tal caso, la primera contraseña es para el comando `sudo` de Mac y la segunda, para el Raspberry Pi.

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> El número de versión de este comando debe coincidir con la versión del paquete de software AWS IoT Greengrass Core.

UNIX-like system

Para transferir los archivos comprimidos desde un equipo a un dispositivo Raspberry Pi central, abra una ventana de terminal en el equipo y ejecute los comandos siguientes:

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> El número de versión de este comando debe coincidir con la versión del paquete de software AWS IoT Greengrass Core.

Raspberry Pi web browser

Si descargó los archivos comprimidos a través del navegador web del Raspberry Pi, dichos archivos seguramente estén en la carpeta `~/Downloads` del dispositivo Pi, como por ejemplo `/home/pi/Downloads`. De lo contrario, los archivos comprimidos deben estar en la carpeta del Pi `~`, como por ejemplo `/home/pi`.

3. En el dispositivo del núcleo de Greengrass, abra una terminal y vaya a la carpeta que contiene el software AWS IoT Greengrass Core y los certificados. Sustituya la *`path-to-transferred-files`* por la ruta a la que transfirió los archivos en el dispositivo del núcleo. Por ejemplo, en un Raspberry Pi, ejecute `cd /home/pi`.

```
cd path-to-transferred-files
```

4. Descomprima el software de AWS IoT Greengrass Core en el dispositivo principal. Ejecute el siguiente comando para descomprimir el archivo de software que ha transferido al dispositivo del núcleo. Este comando usa el argumento `-C /` para crear la carpeta `/greengrass` en la carpeta raíz del dispositivo del núcleo.

```
sudo tar -xzvf greengrass-OS-architecture-1.11.6.tar.gz -C /
```

> **Note**
>
> El número de versión de este comando debe coincidir con la versión del paquete de software AWS IoT Greengrass Core.

5. Mueva los certificados y las claves a la carpeta del software AWS IoT Greengrass Core. Ejecute los siguientes comandos para crear una carpeta de certificados y mover los certificados y las claves a ella. Reemplace *`path-to-transferred-files`* por la ruta a la que transfirió los archivos en el dispositivo del núcleo y reemplace *`certificateId`* por la identificación del certificado en los nombres de los archivos. *`Por ejemplo, en un Raspberry Pi, sustituya path-to-transferred-files`* por **`/home/pi`**.

```
sudo mv path-to-transferred-files/certificateId-certificate.pem.crt /greengrass/
certs
sudo mv path-to-transferred-files/certificateId-public.pem.key /greengrass/certs
sudo mv path-to-transferred-files/certificateId-private.pem.key /greengrass/certs
sudo mv path-to-transferred-files/AmazonRootCA1.pem /greengrass/certs
```

6. El software AWS IoT Greengrass Core utiliza un archivo de configuración que especifica los parámetros del software. Este archivo de configuración especifica las rutas de los archivos de certificados y los puntos de conexión de Nube de AWS a utilizar. En este paso, creará el archivo de configuración del software AWS IoT Greengrass Core para su núcleo. Haga lo siguiente:

   a. Obtenga el nombre de recurso de Amazon (ARN) para el objeto AWS IoT de su núcleo. Haga lo siguiente:

      i. En la consola AWS IoT, en Administrar, en Dispositivos Greengrass, elija Grupos (V1).

      ii. En la página de grupos de Greengrass, elija el grupo que creó anteriormente.

      iii. En Descripción general, seleccione el núcleo de Greengrass.

      iv. En la página de detalles del núcleo, copie el ARN del objeto AWS IoT y guárdelo para usarlo en el archivo de configuración de AWS IoT Greengrass Core.

   b. Obtenga el punto de enlace de datos del dispositivo AWS IoT para su Cuenta de AWS en la región actual. Los dispositivos utilizan este punto de conexión para conectar AWS como objetos AWS IoT. Haga lo siguiente:

      i. En la consola de AWS IoT, elija Configuración.

      ii. En Punto de enlace de datos de dispositivo, copie el punto de conexión y guárdelo para usarlo en el archivo de configuración de AWS IoT Greengrass Core.

   c. El archivo de configuración del software de AWS IoT Greengrass Core. Por ejemplo, puede ejecutar el comando siguiente para usar GNU nano para crear el archivo.

      ```
      sudo nano /greengrass/config/config.json
      ```

      Reemplace el contenido del archivo por el siguiente documento JSON:

      ```
      {
        "coreThing" : {
          "caPath": "AmazonRootCA1.pem",
          "certPath": "certificateId-certificate.pem.crt",
      ```

```
    "keyPath": "certificateId-private.pem.key",
    "thingArn": "arn:aws:iot:region:account-id:thing/MyGreengrassV1Core",
    "iotHost": "device-data-prefix-ats.iot.region.amazonaws.com",
    "ggHost": "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes"
    }
  },
  "managedRespawn": false,
  "crypto": {
    "caPath": "file:///greengrass/certs/AmazonRootCA1.pem",
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-
private.pem.key"
      },
      "IoTCertificate": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-
private.pem.key",
        "certificatePath": "file:///greengrass/certs/certificateId-
certificate.pem.crt"
      }
    }
  }
}
```

A continuación, proceda del modo siguiente:

- Si descargó un certificado de CA raíz de Amazon diferente al de Amazon Root CA 1, sustituya cada instancia de *AmazonRootCA1.pem* por el nombre del archivo de CA raíz de Amazon.
- Reemplace cada instancia de *certificateId* por la identificación del certificado en el nombre de los archivos de certificado y clave.
- Reemplace *arn:aws:iot:region:account-id:thing/MyGreengrassV1Core* por el ARN su objeto del núcleo que guardó anteriormente.
- Reemplace *MyGreengrassV1core* por el nombre su objeto del núcleo.
- Reemplace *device-data-prefix-ats.iot.region.amazonaws.com* por el punto de enlace de datos del dispositivo AWS IoT que guardó anteriormente.

- Sustituya *región* por su Región de AWS.

  Para obtener más información sobre las opciones de configuración que puede especificar en este archivo de configuración, consulte Archivo de configuración de AWS IoT Greengrass Core.

7. Asegúrese de que su dispositivo central esté conectado a internet. A continuación, inice AWS IoT Greengrass en su dispositivo principal.

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

Debería aparecer un mensaje `Greengrass successfully started`. Tome nota del PID.

> **Note**
>
> Para configurar su dispositivo del núcleo para iniciar AWS IoT Greengrass en el arranque del sistema, consulte, consulte the section called “Iniciar Greengrass en el arranque del sistema”.

Puede ejecutar el siguiente comando para confirmar que el software de AWS IoT Greengrass Core (daemon de Greengrass) funciona. Reemplace *número-PID* por su PID:

```
ps aux | grep PID-number
```

Debería ver una entrada para el PID con una ruta al demonio de Greengrass en ejecución (por ejemplo, `/greengrass/ggc/packages/1.11.6/bin/daemon`). Si tiene problemas para iniciar AWS IoT Greengrass, consulte Solución de problemas.

Módulo 3 (primera parte): Funciones de Lambda en AWS IoT Greengrass

Este módulo le muestra cómo crear e implementar una función de Lambda que envía mensajes de MQTT desde su dispositivo principal AWS IoT Greengrass. El módulo describe las configuraciones

de funciones de Lambda , las suscripciones utilizadas para permitir los mensajes de MQTT y las implementaciones en un dispositivo principal.

En el módulo 3 (segunda parte) se analizan las diferencias entre las funciones de Lambda bajo demanda y de larga duración que se ejecutan en el núcleo AWS IoT Greengrass.

Antes de comenzar, asegúrese de que ha completado el Módulo 1 y el Módulo 2 y tiene un dispositivo principal de AWS IoT Greengrass en funcionamiento.

> **Tip**
>
> O bien, para utilizar un script que configure el dispositivo principal automáticamente, consulte the section called “Inicio rápido: configuración del dispositivo Greengrass”. El script también puede crear e implementar la función de Lambda utilizada en este módulo.

Completar este módulo debería tomarle aproximadamente 30 minutos.

Temas

- Crear y empaquetar una función de Lambda
- Configurar una función de Lambda para AWS IoT Greengrass
- Implementar configuraciones de nube en un dispositivo central de Greengrass
- Verificación de la ejecución de la función de Lambda en el dispositivo central

Crear y empaquetar una función de Lambda

La función de Lambda de Python de ejemplo de este módulo utiliza el SDK AWS IoT Greengrass Core de Python para publicar mensajes de MQTT.

En este paso:

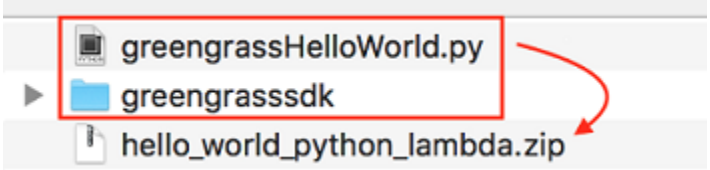
- Descargue el SDK de AWS IoT Greengrass Core para Python en su equipo (no en el dispositivo de AWS IoT Greengrass).
- Creará un paquete de implementación de funciones de Lambda que contiene código de característica y dependencias.
- Utilizará la consola de Lambda para crear una función de Lambda y cargar el paquete de implementación.
- Publicará una versión de la función de Lambda y creará un alias que apunte a la versión.

Para completar este módulo, Python 3.7 debe estar instalado en su dispositivo principal.

1. Desde la página de descargas del Core SDK AWS IoT Greengrass, descargue el SDK AWS IoT Greengrass básico para Python en su ordenador.
2. Descomprimir el paquete descargado para obtener el código de la función de Lambda y el SDK.

   La función de Lambda de este módulo utiliza:

   - El archivo `greengrassHelloWorld.py` en `examples\HelloWorld`. Este es el código de la función de Lambda. Cada cinco segundos, la característica publica uno de los dos mensajes posibles en el tema de `hello/world`.
   - La carpeta `greengrasssdk`. Este es el SDK.
3. Copie la carpeta `greengrasssdk` en la carpeta `HelloWorld` que contiene `greengrassHelloWorld.py`.
4. Para crear el paquete de implementación de la función de Lambda, guarde `greengrassHelloWorld.py` y la carpeta `greengrasssdk` en un archivo `zip` comprimido denominado `hello_world_python_lambda.zip`. El archivo `py` y la carpeta `greengrasssdk` deben estar en la raíz del directorio.



   En sistemas de tipo UNIX (incluido el terminal de Mac), puede utilizar el siguiente comando para empaquetar el archivo y la carpeta:

```
zip -r hello_world_python_lambda.zip greengrasssdk greengrassHelloWorld.py
```

> **Note**
>
> En función de la distribución, es posible que tenga que instalar `zip` primero (por ejemplo, ejecutando `sudo apt-get install zip`). El comando de instalación podría ser diferente en su distribución.

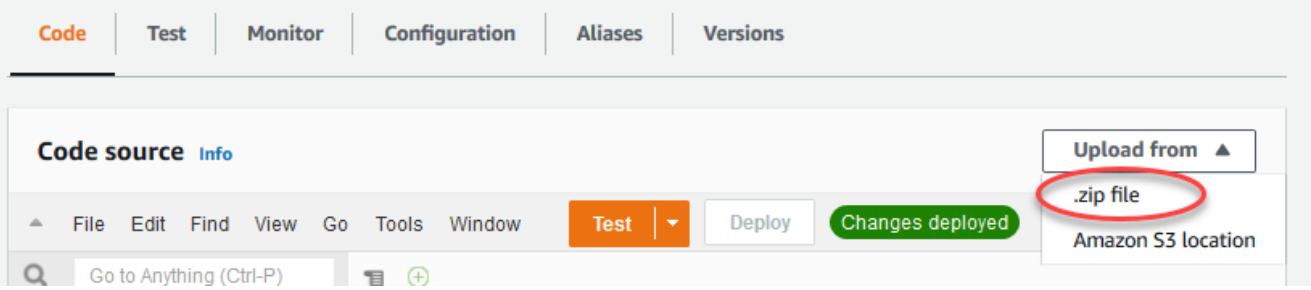
Ya está listo para crear la función de Lambda; y cargar el paquete de implementación.

5. Abra la consola de Lambda; y elija Crear función.
6. Elija Author from scratch (Crear desde cero).
7. Asigne el nombre **`Greengrass_HelloWorld`** a la característica y establezca los demás campos como se indica a continuación:

   - En Runtime (Tiempo de ejecución), elija Python 3.7.
   - En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol.

   Elija Crear función.
8. Cargue su paquete de implementación de la función de Lambda:

   a. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip..



   b. Seleccione Cargar y, a continuación, elija su paquete de implementación `hello_world_python_lambda.zip`. A continuación, elija Guardar.
   c. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

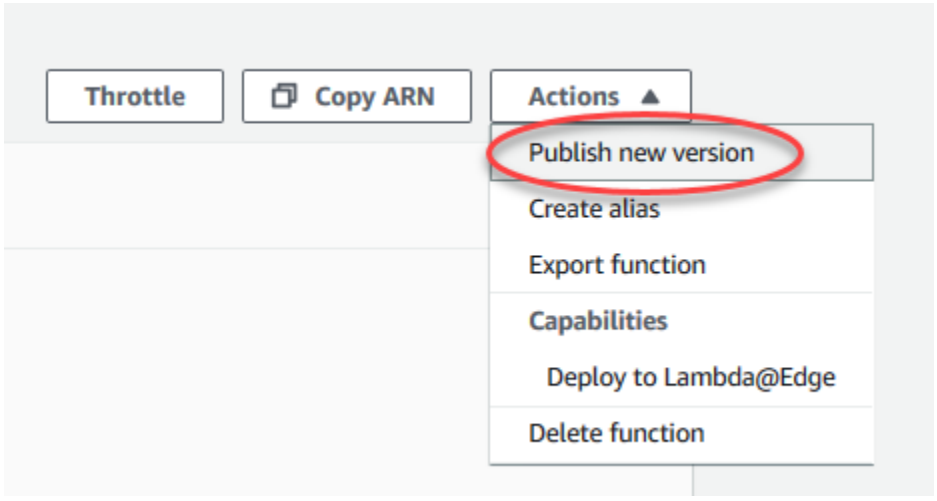
      - En Runtime (Tiempo de ejecución), elija Python 3.7.
      - En Handler (Controlador), escriba **`greengrassHelloWorld.function_handler`**.

d. Seleccione Save.

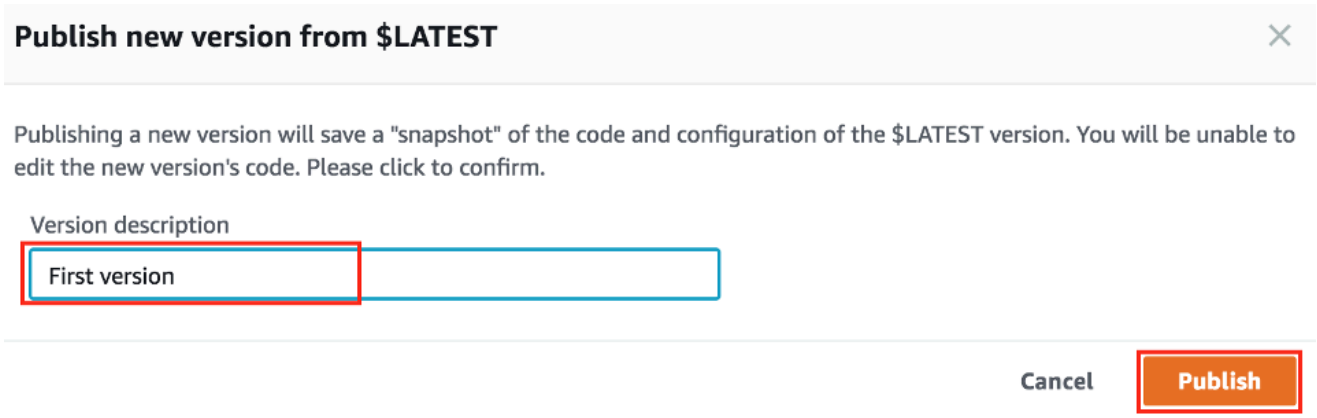
> **Note**
>
> El botón de prueba de la consola de AWS Lambda no funciona con esta función. El SDK AWS IoT Greengrass Core no contiene los módulos necesarios para ejecutar las funciones de Lambda de Greengrass de forma independiente en la consola AWS Lambda. Estos módulos (por ejemplo, `greengrass_common`) se suministran a las funciones una vez desplegados en el núcleo de Greengrass.

9. Publica la función de Lambda:

a. En el menú Acciones de la parte superior de la página, elija Publicar nueva versión.



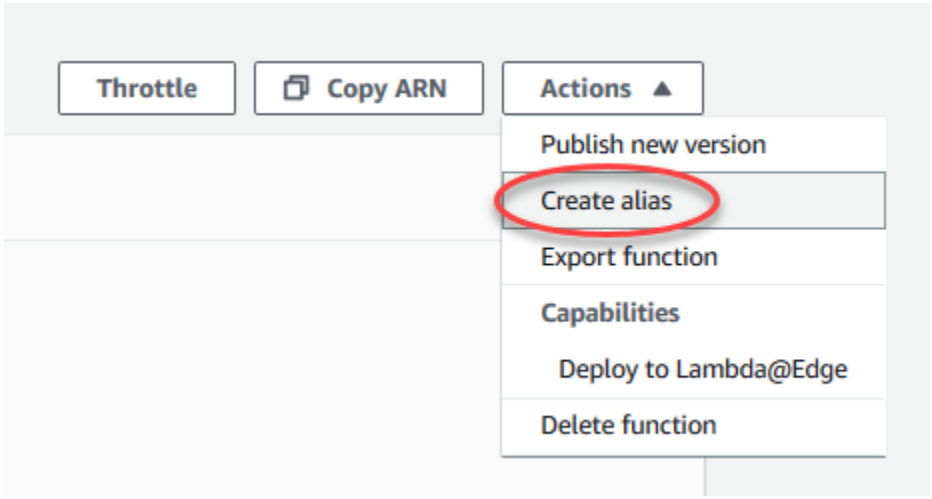
b. En Version description (Descripción de versión), escriba **`First version`** y, a continuación, elija Publish (Publicar).



10. Cree un alias para la versión de la función de Lambda:

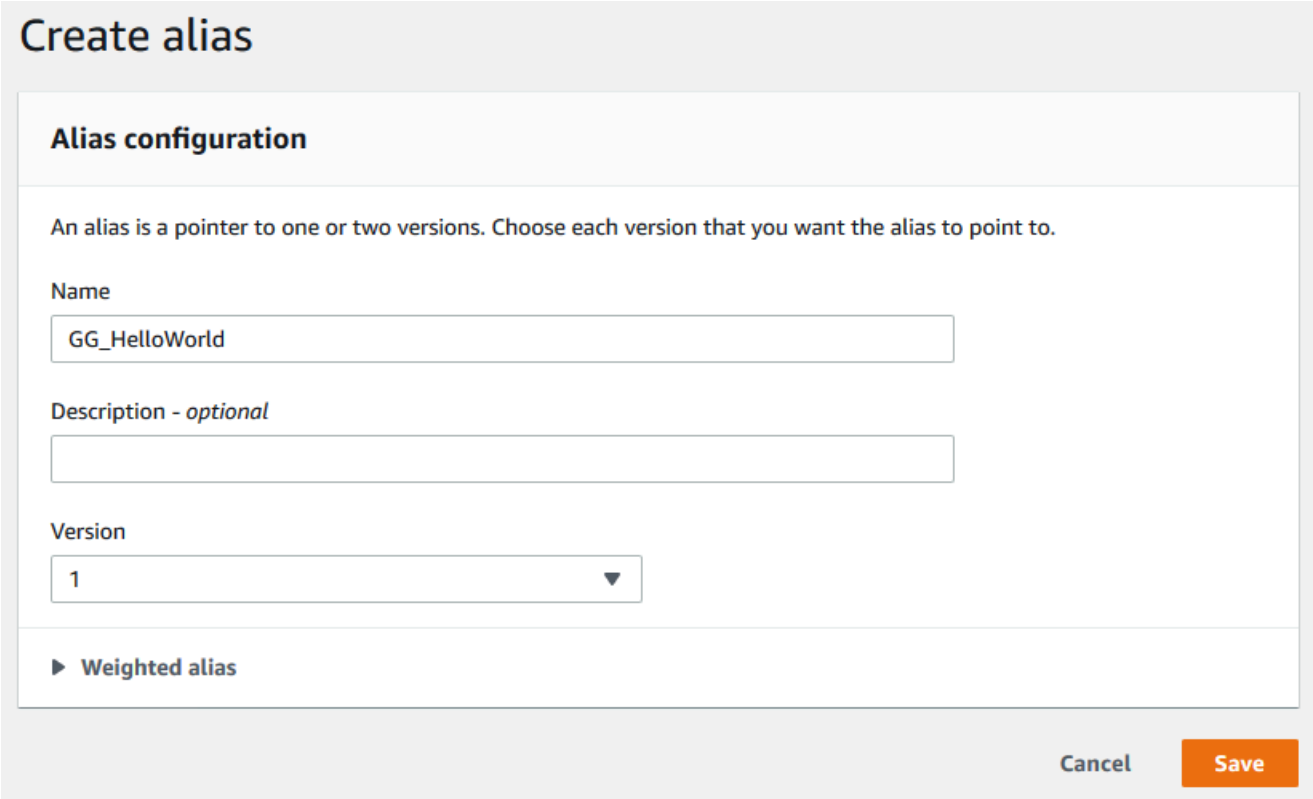
> **Note**
>
> Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

a. En el menú Acciones de la parte superior de la página, seleccione Crear alias.



b. Denomine al alias **`GG_HelloWorld`**, establezca la versión en **`1`** (que corresponde con la versión que acaba de publicar) y, a continuación, elija Guardar.

> **Note**
>
> AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST.



Configurar una función de Lambda para AWS IoT Greengrass

Ahora está listo para configurar la función de Lambda para AWS IoT Greengrass.

En este paso:

- Utilizará la consola de AWS IoT para añadir la función de Lambda al grupo de Greengrass.
- Configurará los ajustes específicos del grupo para la función de Lambda.
- Añadirá una suscripción al grupo que permita a la función de Lambda publicar mensajes de MQTT en AWS IoT.
- Configurará las opciones de registro local para el grupo.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. En Grupos de Greengrass, elija el grupo que creó en el módulo 2.
3. En la página de configuración del grupo, elija la pestaña Funciones de Lambda y, a continuación, desplácese hacia abajo hasta la sección Mis funciones de Lambda y elija Añadir función de Lambda.
4. Seleccione el nombre de la función de Lambda que creó en el paso anterior (Greengrass_HelloWorld, no el nombre del alias).
5. Para la versión, elija Alias: GG_HelloWorld.
6. En la sección de configuración de la función de Lambda, realice los siguientes cambios:
   - Establezca el usuario y el grupo del sistema en Utilizar grupo por defecto.
   - Establezca la creación de contenedores de la función de Lambda en Usar grupo por defecto.
   - Configure el Timeout (Tiempo de espera) en 25 segundos. Esta función de Lambda está inactiva durante 5 segundos antes de cada invocación.
   - En Ancladas, elija Verdadero

   > **Note**
   >
   > Una función de Lambda de larga duración (o anclada) se inicia automáticamente después de arrancar AWS IoT Greengrass y sigue ejecutándose en su propio contenedor. Esto contrasta con una función de Lambda bajo demanda, que se inicia cuando se la invoca y se detiene cuando no quedan tareas que ejecutar. Para obtener más información, consulte the section called “Configuración del ciclo de vida”.

7. Elija Agregar función de Lambda para guardar los cambios. Para obtener información sobre las propiedades de la función de Lambda, consulte the section called “Control de la ejecución de la función de Lambda de Greengrass”.

   A continuación, cree una suscripción que permita a la función de Lambda enviar mensajes MQTT a AWS IoT Core.

Una función de Lambda de Greengrass puede intercambiar mensajes MQTT con:

- Dispositivos del grupo de Greengrass
- Conectores en el grupo.
- Otras funciones de Lambda del grupo.
- AWS IoT Core.
- El servicio de sombra local Para obtener más información, consulte the section called “Módulo 5: Interacción con sombras de dispositivos”.

El grupo utiliza las suscripciones para controlar la forma en que estas entidades se pueden comunicar entre sí. Las suscripciones proporcionan interacciones predecibles y una capa de seguridad.

Una suscripción se compone de un origen, un destino y un tema. El origen es el autor del mensaje. El destino es el destinatario del mensaje. El tema permite filtrar los datos que se envían desde el origen hasta el destino. El origen o el destino pueden ser un dispositivo de Greengrass, una función de Lambda, un conector, una sombra de dispositivo o AWS IoT Core.

> **Note**
>
> La suscripción está dirigida, ya que los mensajes fluyen en una dirección específica: del origen al destino. Para permitir la comunicación bidireccional, debe configurar dos suscripciones.

> **Note**
>
> Actualmente, el filtro de temas de suscripción no permite más de un carácter + en un tema. El filtro de temas solo permite un carácter # al final de un tema.

Dado que la función de Lambda `Greengrass_HelloWorld` solo envía mensajes al tema `hello/world` en AWS IoT Core , solo tiene que crear una suscripción de la función de Lambda a AWS IoT Core. Cree esto en el siguiente paso.

8. En la página de configuración del grupo, elija la pestaña Suscripciones y, a continuación, elija Añadir suscripción.

   Para ver un ejemplo que muestra cómo crear una suscripción mediante la AWS CLI, consulte create-subscription-definition en la Referencia de comandos de la AWS CLI.

9. En el Tipo de origen, elija Función de Lambda y, para el Origen, elija Greengrass_HelloWorld.
10. Para el Tipo de destino, elija Servicio y, para el objetivo, seleccione Nube de IoT.
11. En Filtro por temas, introduzca **`hello/world`** y, a continuación, seleccione Crear suscripción.
12. Defina la configuración de registro del grupo. En este tutorial, debe configurar los componentes del sistema de AWS IoT Greengrass y las funciones de Lambda definidas por el usuario para que escriban los registros en el sistema de archivos del dispositivo del núcleo.

    a. En la página de configuración del grupo, elija la pestaña Registros.

    b. En la sección Configuración de registros locales, elija Editar.

    c. En el cuadro de diálogo Editar la configuración de los registros locales, mantenga los valores predeterminados tanto para los niveles de registro como para los tamaños de almacenamiento y, a continuación, seleccione Guardar.

    Puede usar registros para solucionar cualquier problema que pueda surgir al ejecutar este tutorial. Durante la solución de problemas, puede cambiar temporalmente el nivel del registro a Depuración. Para obtener más información, consulte the section called “Acceso a los registros del sistema de archivos”.

13. Si Java 8 Runtime no está instalado en el dispositivo principal, debe instalarlo o desactivar el administrador de secuencias.

    > **Note**
    >
    > En este tutorial no se utiliza el administrador de secuencias, pero sí el flujo de trabajo Creación predeterminada de un grupo, que habilita el administrador de secuencias de forma predeterminada. Si el administrador de secuencias está habilitado pero Java 8 no está instalado, se produce un error en la implementación del grupo. Para obtener más información, consulte los requisitos del administrador de secuencias.

    Para desactivar el administrador de secuencias:

a. En la página de configuración del grupo, elija la pestaña Funciones de lambda.

b. En la sección Funciones de Lambda del sistema, seleccione Administrador de secuencias y luego elija Editar.

c. Elija Deshabilitar y, a continuación, Guardar.

Implementar configuraciones de nube en un dispositivo central de Greengrass

1. Asegúrese de que el dispositivo central esté conectado a internet. Por ejemplo, intente navegar correctamente a una página web.
2. Asegúrese de que el demonio de Greengrass se esté ejecutando en su dispositivo del núcleo. En el terminal de su dispositivo central, ejecute los siguientes comandos para comprobar si el daemon se está ejecutando e inícielo, si es necesario.

   a. Para comprobar si el daemon está en ejecución:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, el daemon está en ejecución.

   b. Iniciar el daemon:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

   Ya está listo para implementar la función de Lambda y las configuraciones de suscripciones en el dispositivo central de Greengrass.
3. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
4. En Grupos de Greengrass, elija el grupo que creó en el módulo 2.
5. En la página de configuración de grupo, elija Implementar.
6. En la pestaña Funciones de Lambda, en la sección Funciones de Lambda del sistema, elija Detector IP.

7. Elija Editar y seleccione Detectar y anular automáticamente los puntos de conexión del agente de MQTT. Esto permite a los dispositivos adquirir automáticamente la información de conexión del dispositivo principal, como la dirección IP, el DNS y el número de puerto. Se recomienda la detección automática, pero AWS IoT Greengrass también es compatible con puntos de conexión especificados manualmente. Solo se le solicitará el método de detección la primera vez que se implemente el grupo.

La primera implementación puede tardar unos minutos. Cuando termine la implementación, debería ver el mensaje Successfully completed (Realizado correctamente) en la columna Status (Estado) de la página Deployments (Implementaciones):

> **Note**
>
> El estado de implementación aparece también en la parte superior de la página, bajo el nombre del grupo.

Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

Verificación de la ejecución de la función de Lambda en el dispositivo central

1. En el panel de navegación de la consola AWS IoT, en Probar, elija el cliente de prueba MQTT.
2. Elija la pestaña Suscribirse al tema.
3. Escriba **`hello/world`** en el filtro de temas y amplíe la configuración adicional.
4. Introduzca la información que aparece en cada uno de los campos siguientes:
   - En Quality of Service (Calidad del servicio), seleccione 0.
   - En MQTT payload display (Visualización de la carga de MQTT), seleccione Display payloads as strings (Mostrar cargas como cadenas).
5. Elija Subscribe.

Suponiendo que la función de Lambda se esté ejecutando en el dispositivo, publica mensajes similares al siguiente en el tema `hello/world`:

Aunque la función de Lambda continúe enviando mensajes MQTT al tema `hello/world`, no detiene el daemon de AWS IoT Greengrass. El resto de módulos se escriben partiendo del supuesto de que está en ejecución.

Puede eliminar la función y la suscripción del grupo:

- En la página de configuración de grupos, en la pestaña Funciones de lambda, seleccione la función de Lambda que desea eliminar y elija Eliminar.
- En la página de configuración del grupo, seleccione la pestaña Suscripciones y luego elija Suprimir.

La función y la suscripción se eliminan del dispositivo central durante la siguiente implementación del grupo.

Módulo 3 (segunda parte): Funciones de Lambda en AWS IoT Greengrass

En este módulo, se explican las diferencias entre las funciones de Lambda bajo demanda y de larga duración que se ejecutan en el núcleo AWS IoT Greengrass.

Antes de comenzar, ejecute el script de configuración de dispositivos de Greengrass o asegúrese de haber completado el módulo 1, el módulo 2 y el módulo 3 (Parte 1).

Completar este módulo debería tomarle aproximadamente 30 minutos.

Temas

- Creación y empaquetado de la función de Lambda

- Configuración de funciones de Lambra de larga duración en AWS IoT Greengrass
- Prueba de funciones de Lambda de larga duración
- Pruebe funciones de Lambda bajo demanda

Creación y empaquetado de la función de Lambda

En este paso:

- Creará un paquete de implementación de funciones de Lambda que contiene código de característica y dependencias.
- Utilizará la consola de Lambda para crear una función de Lambda y cargar el paquete de implementación.
- Publicará una versión de la función de Lambda y creará un alias que apunte a la versión.

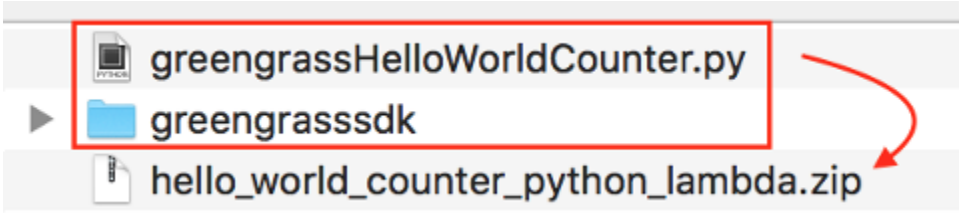
1. En su equipo, vaya al SDK AWS IoT Greengrass Core para Pyton que descargó y extrajo en the section called “Crear y empaquetar una función de Lambda” en el módulo 3-1.

   La función de Lambda de este módulo utiliza:

   - El archivo `greengrassHelloWorldCounter.py` en `examples\HelloWorldCounter`. Este es el código de la función de Lambda.
   - La carpeta `greengrasssdk`. Este es el SDK.
2. Creación de un paquete de implementación de la función de Lambda:

   a. Copie la carpeta `greengrasssdk` en la carpeta `HelloWorldCounter` que contiene `greengrassHelloWorldCounter.py`.

   b. Guarde `greengrassHelloWorldCounter.py` y la carpeta `greengrasssdk` en un archivo `zip` llamado `hello_world_counter_python_lambda.zip`. El archivo py y la carpeta `greengrasssdk` deben estar en la raíz del directorio.



En sistemas de tipo UNIX (incluido el terminal de Mac) que tengan `zip` instalado, puede utilizar el siguiente comando para empaquetar el archivo y la carpeta:

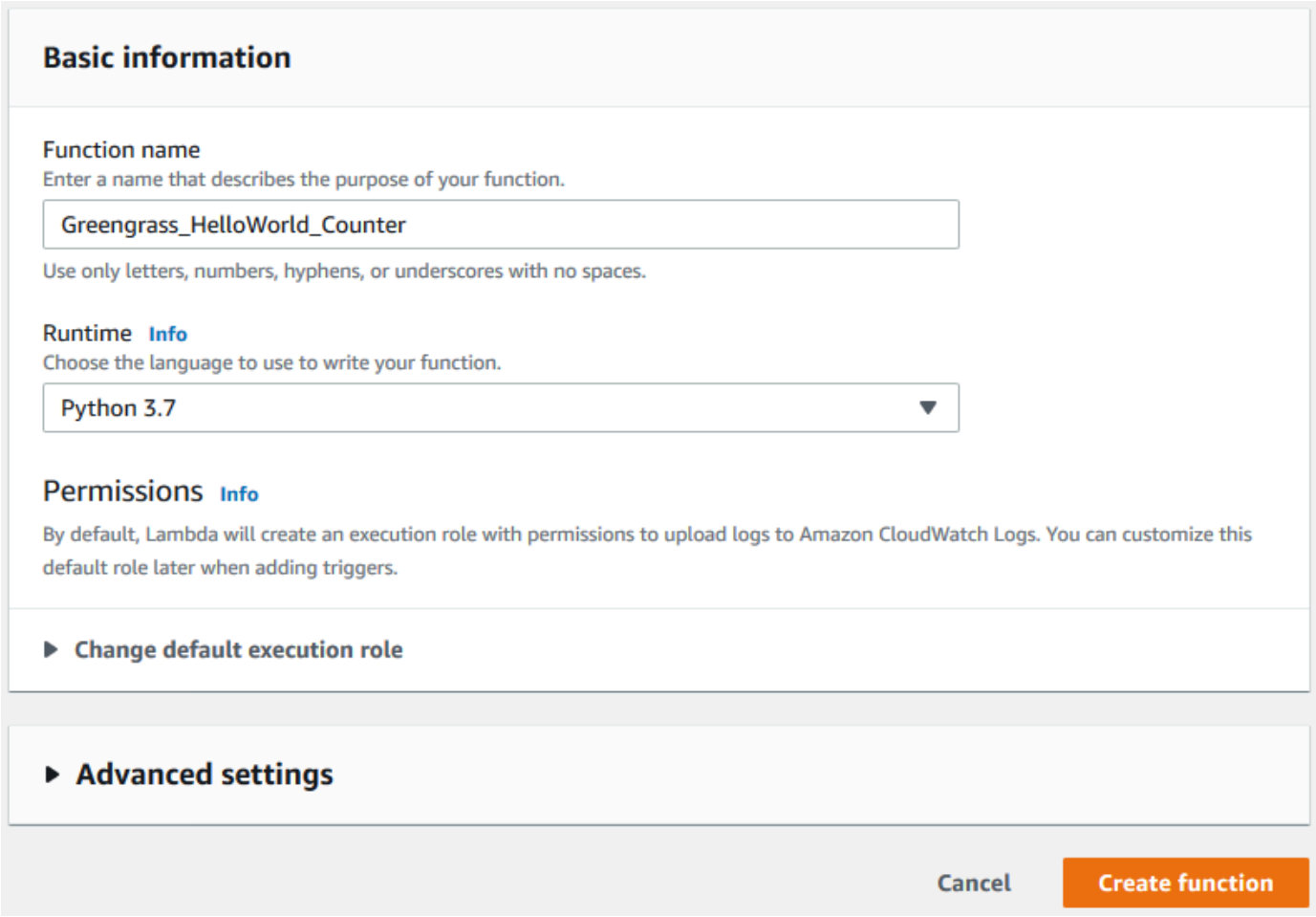
```
zip -r hello_world_counter_python_lambda.zip greengrasssdk
 greengrassHelloWorldCounter.py
```

Ya está listo para crear la función de Lambda; y cargar el paquete de implementación.

3. Abra la consola de Lambda; y elija Crear función.
4. Elija Author from scratch (Crear desde cero).
5. Asigne el nombre **`Greengrass_HelloWorld_Counter`** a la función y establezca los demás campos como se indica a continuación:

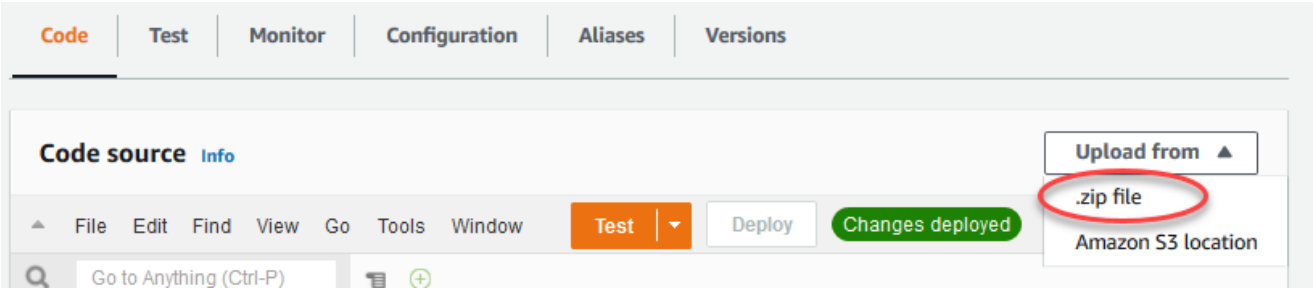
   - En Runtime (Tiempo de ejecución), elija Python 3.7.
   - En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol. O bien puede reutilizar el rol que creó en el módulo 3-1.

   Elija Crear función.



6. Cargue su paquete de implementación de la función de Lambda.

   a. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip.



   b. Seleccione Cargar y, a continuación, elija su paquete de implementación `hello_world_counter_python_lambda.zip`. A continuación, elija Save (Guardar).

   c. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

      - En Runtime (Tiempo de ejecución), elija Python 3.7.

- En Handler (Controlador), escriba **`greengrassHelloWorldCounter.function_handler`**.

d. Seleccione Save.

> **Note**
>
> El botón de prueba de la consola de AWS Lambda no funciona con esta función. El SDK AWS IoT Greengrass Core no contiene los módulos necesarios para ejecutar las funciones de Lambda de Greengrass de forma independiente en la consola AWS Lambda. Estos módulos (por ejemplo, `greengrass_common`) se suministran a las funciones una vez desplegados en el núcleo de Greengrass.

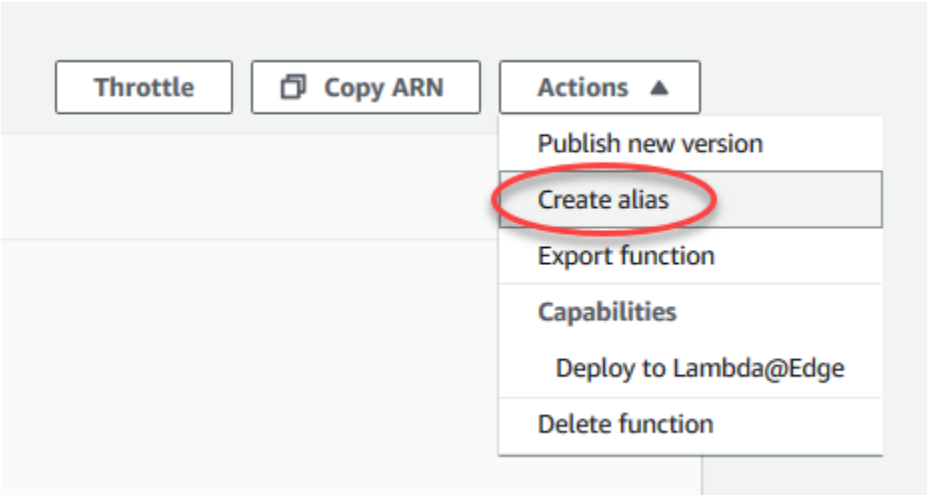
7. Publicar la primera versión de la función.

   a. En el menú Acciones de la parte superior de la página, elija Publicar nueva versión. En Versión description (Descripción de versión), escriba **`First version`**.

   b. Elija Publish.

8. Cree un alias para la versión de la función.

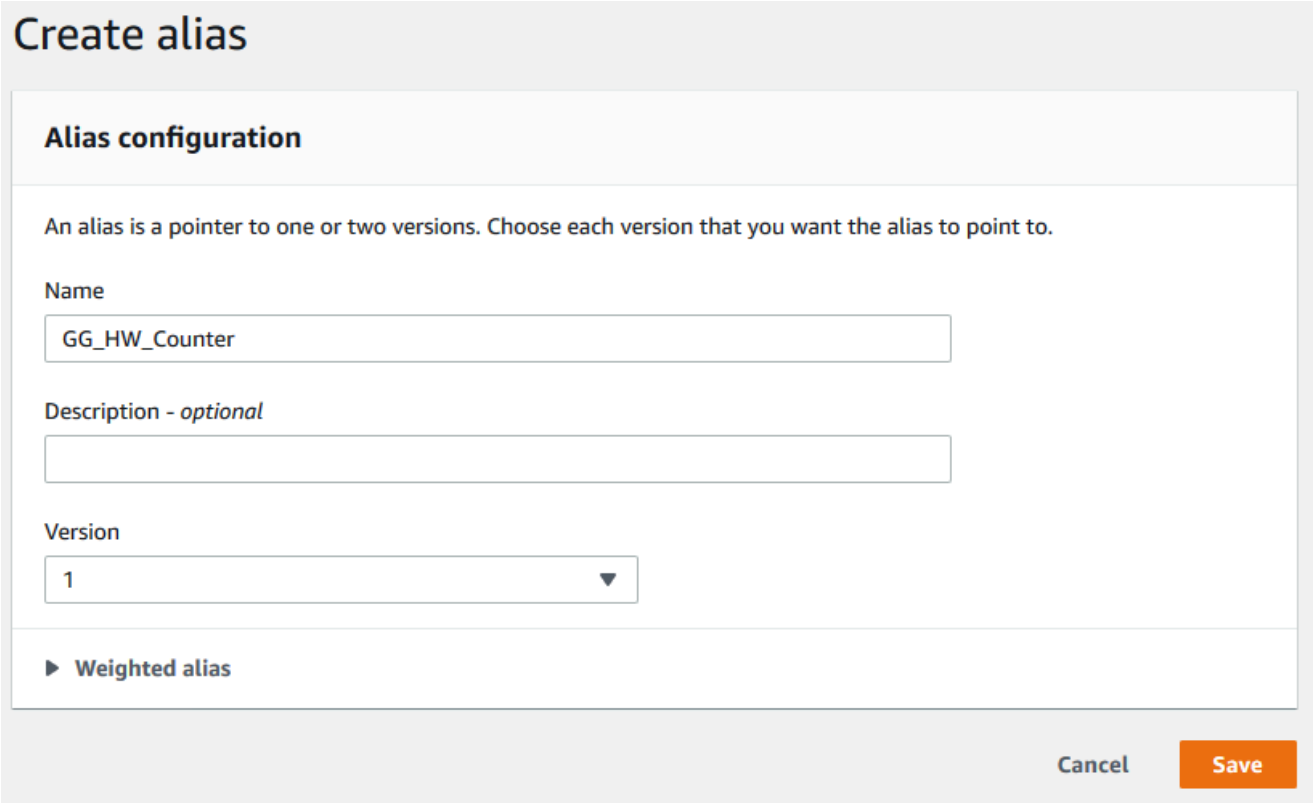
   a. En el menú Acciones de la parte superior de la página, seleccione Crear alias.



   b. En Name (Nombre), ingrese **`GG_HW_Counter`**.

   c. En Version (Versión), elija 1.

   d. Seleccione Save.



Los alias crean una única entidad para su función de Lambda a la que los dispositivos de Greengrass pueden suscribirse. De esta manera, no tendrá que actualizar las suscripciones con nuevos números de versión de funciones de Lambda cada vez que se modifique la función.

Configuración de funciones de Lambra de larga duración en AWS IoT Greengrass

Ahora está listo para configurar la función de Lambda para AWS IoT Greengrass.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. En Grupos de Greengrass, elija el grupo que creó en el módulo 2.
3. En la página de configuración del grupo, elija la pestaña Funciones de Lambda y, a continuación, en Mis funciones de Lambda, seleccione Añadir.
4. Para la función de lambda, elija Greengrass_HelloWorld_Counter.
5. Para la versión de la función de Lambda, elija el alias de la versión que publicó.

6. Para Tiempo de espera (segundos), introduzca **25**. Esta función de Lambda está inactiva durante 20 segundos antes de cada invocación.
7. En Ancladas, elija Verdadero
8. Mantenga los valores predeterminados para todos los demás campos y elija Agregar función de Lambda.

Prueba de funciones de Lambda de larga duración

Una función de Lambda de larga duración se inicia automáticamente cuando se inicia el núcleo AWS IoT Greengrass y se ejecuta en un único contenedor (o entorno aislado). Las variables y la lógica de procesamiento previo que se definen fuera del controlador de la función se conservan para cada invocación del controlador de la función. Las diversas invocaciones del controlador de la función se ponen en cola hasta que se hayan ejecutado las invocaciones anteriores.

El código `greengrassHelloWorldCounter.py` utilizado en este módulo define una variable `my_counter` fuera del controlador de la función.

> **Note**
>
> Puede ver el código en la consola AWS Lambda o en el SDK AWS IoT Greengrass Core para Python en GitHub.

En este paso, puede crear suscripciones que permitan a la función de Lambda y a AWS IoT intercambiar mensajes MQTT. A continuación, implementará el grupo y probará la función.

1. En la página de configuración de grupo, elija Suscripciones y, a continuación, elija Agregar.
2. En Tipo de origen, elija Función de lambda y, a continuación, Greengrass_HelloWorld_Counter.
3. En Tipo de destino, elija Servicio y, a continuación, Nube de IoT.
4. En Filtro de temas, escriba **`hello/world/counter`**.
5. Elija Crear una suscripción.

   Esta única suscripción va en una sola dirección: desde la función de Lambda `Greengrass_HelloWorld_Counter` hasta AWS IoT. Para invocar o activar esta función de Lambda desde la nube, deberá crear una suscripción en la dirección contraria.

6. Siga los pasos del 1 al 5 para añadir otra suscripción que utilice los siguientes valores. Esta suscripción permite que la función de Lambda reciba mensajes de AWS IoT. Esta suscripción se utiliza cuando se envía un mensaje desde la consola de AWS IoT que invoca la función.

   - Para el origen, elija Servicios y después Nube de IoT.
   - Para el destino, elija Funciones de Lambda y después Greengrass_HelloWorld_Counter.
   - En el filtro de temas, escriba **`hello/world/counter/trigger`**.

   La extensión `/trigger` se utiliza en este filtro de temas porque ha creado dos suscripciones y no deben interferir entre sí.
7. Asegúrese de que el daemon de Greengrass esté en ejecución, tal y como se describe en Implementación de configuraciones de nube en un dispositivo central.
8. En la página de configuración de grupo, elija Implementar.
9. Una vez que se haya completado la implementación, vuelva a la página de inicio de la consola AWS IoT y seleccione Probar.
10. Configure los campos siguientes:

    - Para Subscription topic (Tema de suscripción), escriba **`hello/world/counter`**.
    - En Quality of Service (Calidad del servicio), seleccione 0.
    - En MQTT payload display (Visualización de la carga de MQTT), seleccione Display payloads as strings (Mostrar cargas como cadenas).
11. Elija Subscribe.

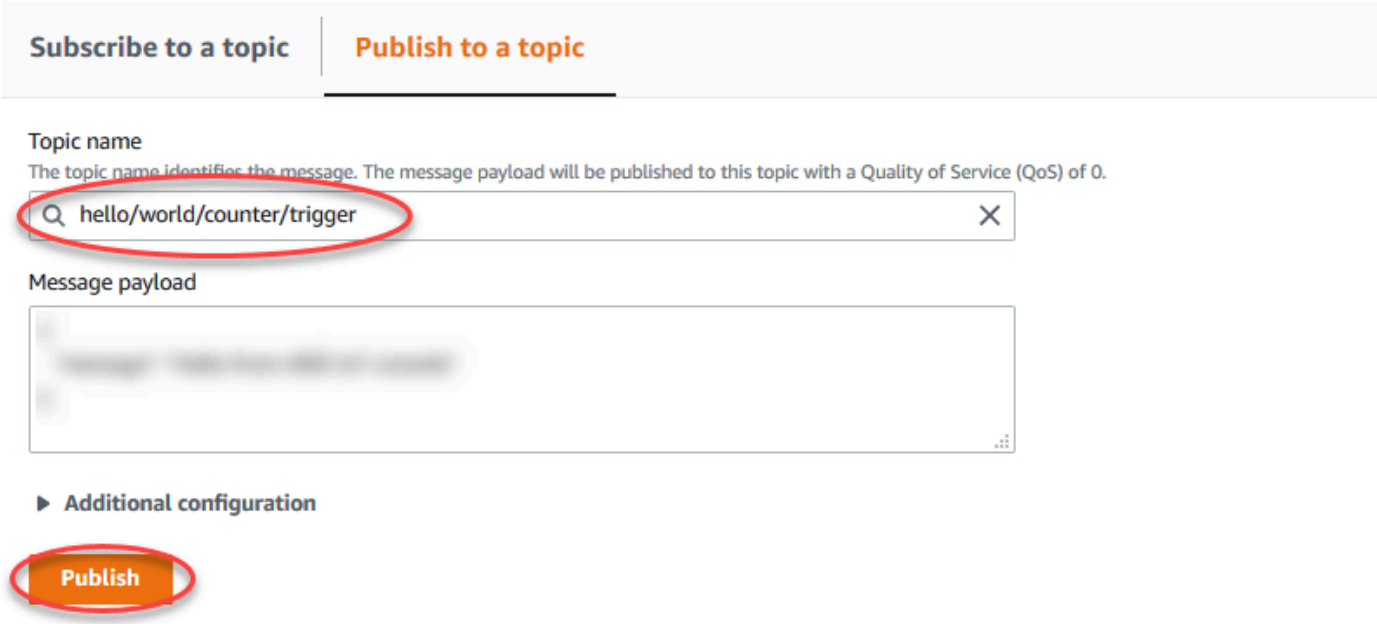
    A diferencia de la primera parte de este módulo, no debería ver ningún mensaje después de suscribirse a `hello/world/counter`. Esto se debe a que el código de `greengrassHelloWorldCounter.py` que se publica en el tema `hello/world/counter` está dentro del controlador de la función, que únicamente se ejecuta cuando se invoca la función.

    En este módulo, configuró la función de Lambda `Greengrass_HelloWorld_Counter` para que se invocara al recibir un mensaje de MQTT sobre el tema `hello/world/counter/trigger`.

    La suscripción que tiene como origen Greengrass_HelloWorld_Counter y como destino IoT Cloud permite que la función envíe mensajes a AWS IoT sobre el tema `hello/world/counter`. La suscripción que tiene como origen IoT Cloud y como destino

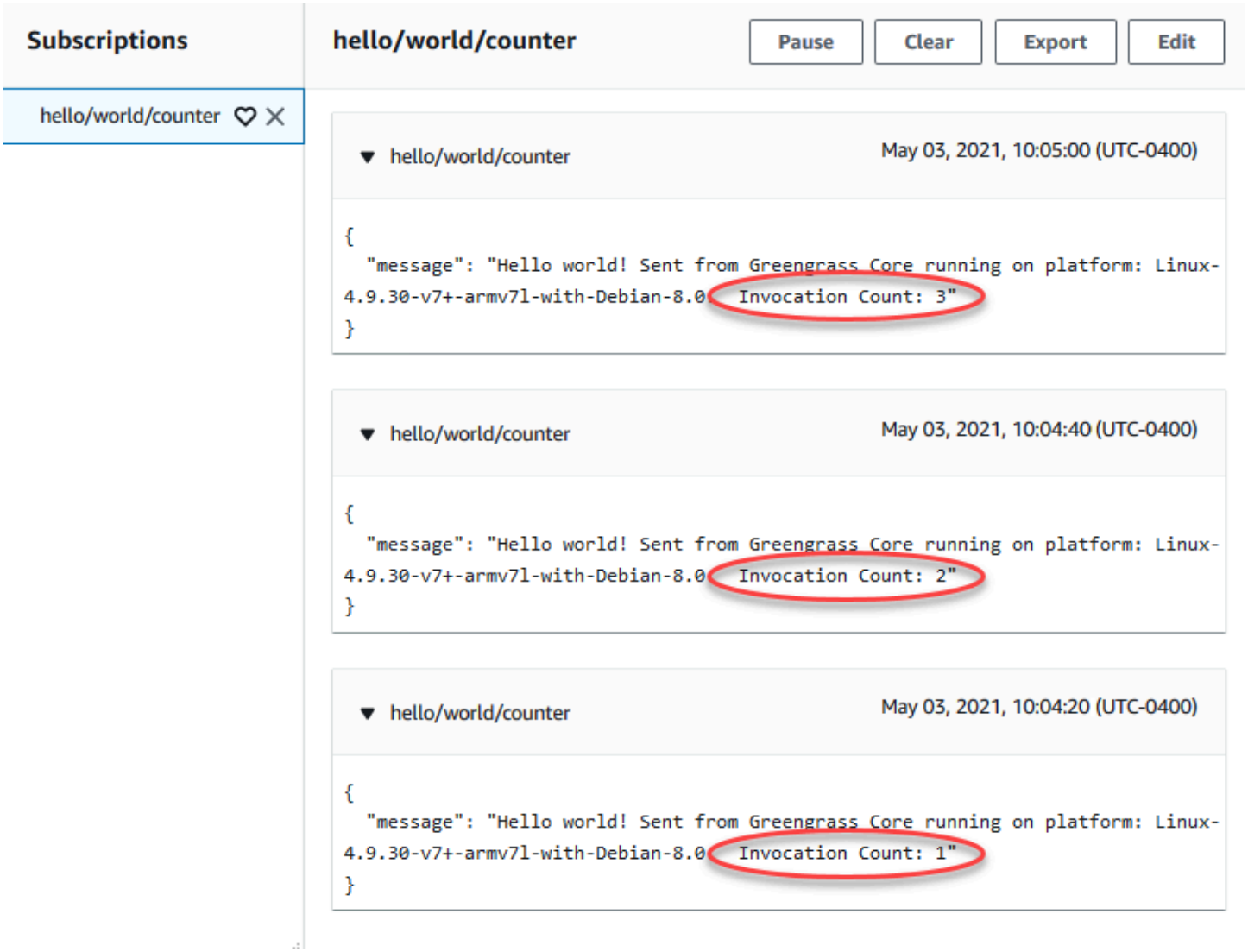
Greengrass_HelloWorld_Counter permite que AWS IoT envíe mensajes a la función sobre el tema `hello/world/counter/trigger`.

12. Para probar el ciclo de vida de larga duración, invoque la función de Lambda publicando un mensaje en el tema `hello/world/counter/trigger`. Puede utilizar el mensaje predeterminado.



> **Note**
>
> La función `Greengrass_HelloWorld_Counter` omite el contenido de los mensajes recibidos. Simplemente ejecuta el código de `function_handler`, que envía un mensaje al tema `hello/world/counter`. Puede consultar este código en la página SDK AWS IoT Greengrass Core para Python en GitHub.

Cada vez que se publica un mensaje en el tema `hello/world/counter/trigger`, la variable `my_counter` se incrementa. Este recuento de invocaciones se muestra en los mensajes enviados de la función de Lambda. Como el controlador de la característica contiene un ciclo de suspensión de 20 segundos (`time.sleep(20)`), si el controlador se activa repetidamente, las respuestas procedentes del núcleo AWS IoT Greengrass se pondrán en cola.

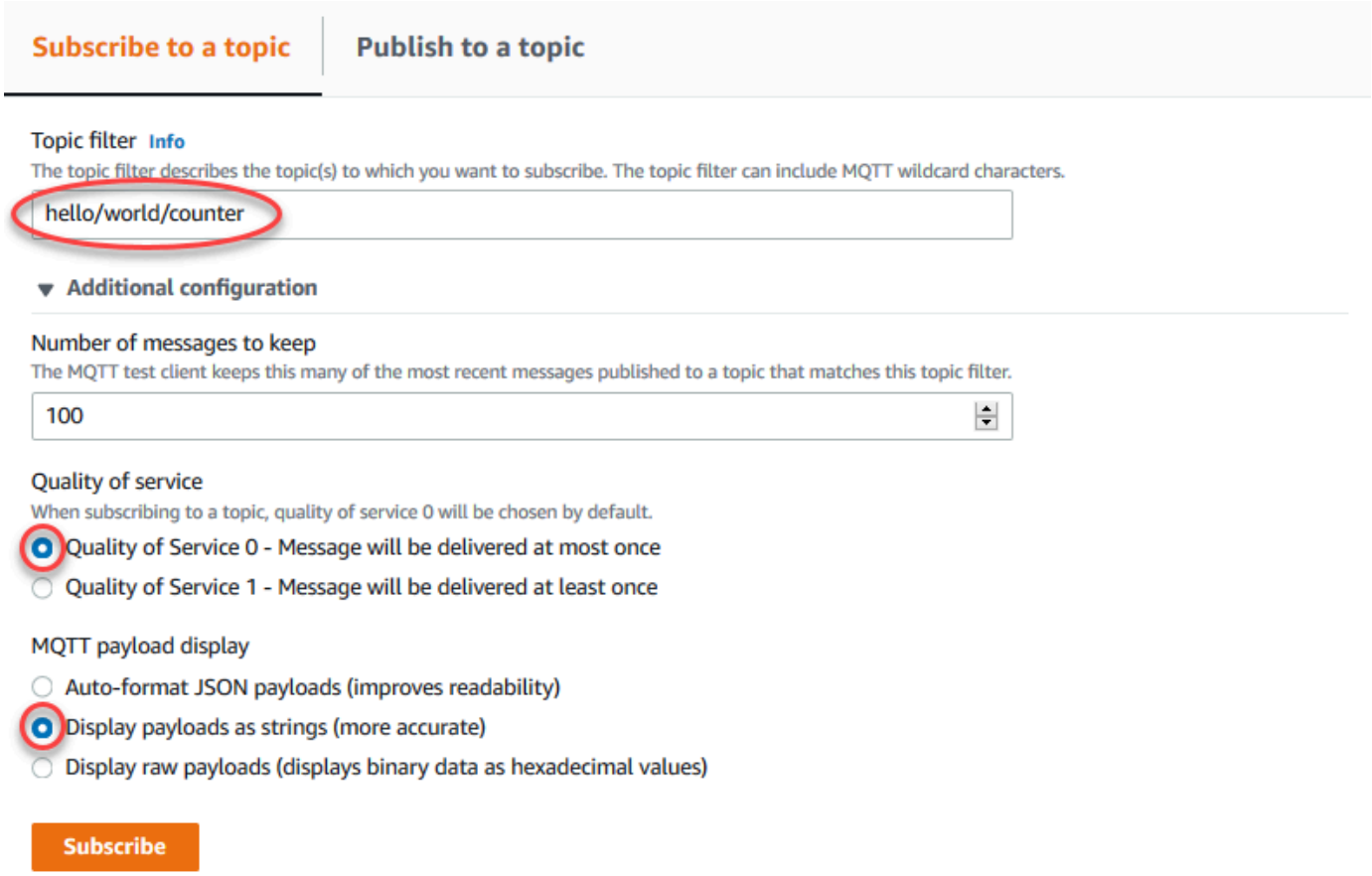


Pruebe funciones de Lambda bajo demanda

Una función de Lambda bajo demanda tiene una funcionalidad similar a una función de AWS Lambda basada en la nube. Es posible ejecutar en paralelo varias invocaciones de una función de Lambda bajo demanda. Cuando se invoca la función de Lambda, se crea un contenedor independiente para procesar las invocaciones o, si los recursos lo permiten, se reutiliza un contenedor existente. No se conserva ninguna variable o procesamiento previo que se defina fuera del controlador de la función cuando se crean contenedores.

1. En la página de configuración del grupo, elija la pestaña Funciones de lambda.
2. En Mis funciones de Lambda, elija la función de Lambda Greengrass_HelloWorld_Counter.
3. En la página de detalles Greengrass_HelloWorld_Counter, elija Editar.
4. En Ancladas, elija Falso y, a continuación, seleccione Guardar.

5. En la página de configuración de grupo, elija Implementar.
6. Una vez que se haya completado la implementación, vuelva a la página de inicio de la consola AWS IoT y seleccione Probar.
7. Configure los campos siguientes:
   - Para Subscription topic (Tema de suscripción), escriba **`hello/world/counter`**.
   - En Quality of Service (Calidad del servicio), seleccione 0.
   - En MQTT payload display (Visualización de la carga de MQTT), seleccione Display payloads as strings (Mostrar cargas como cadenas).

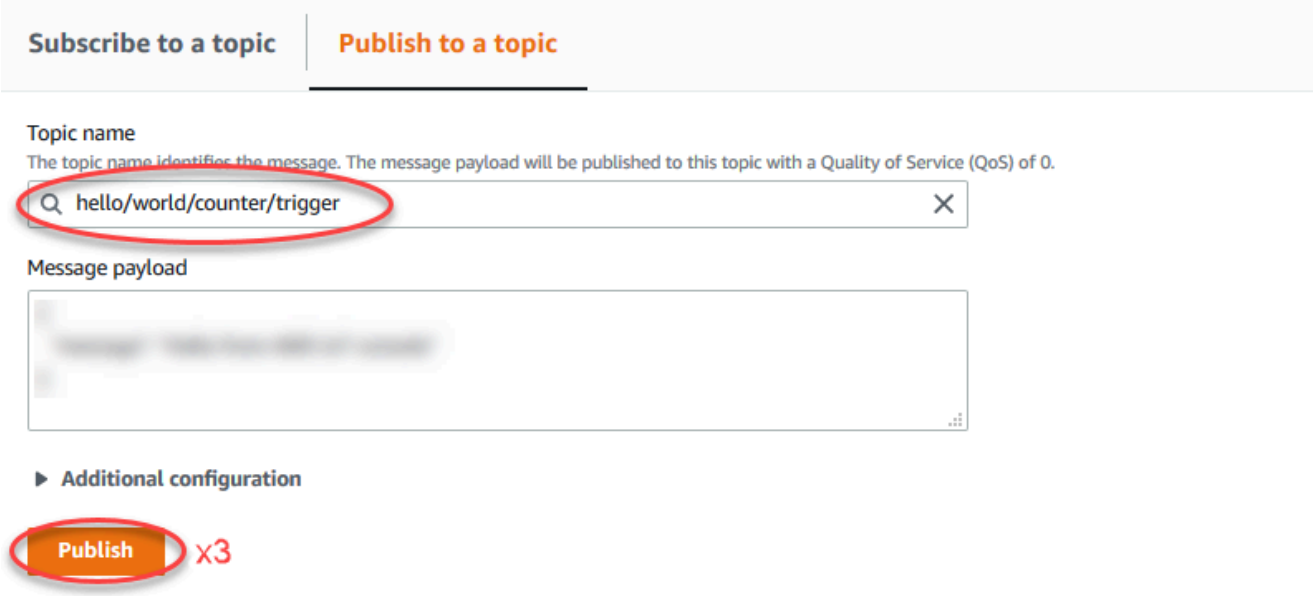


8. Elija Subscribe.

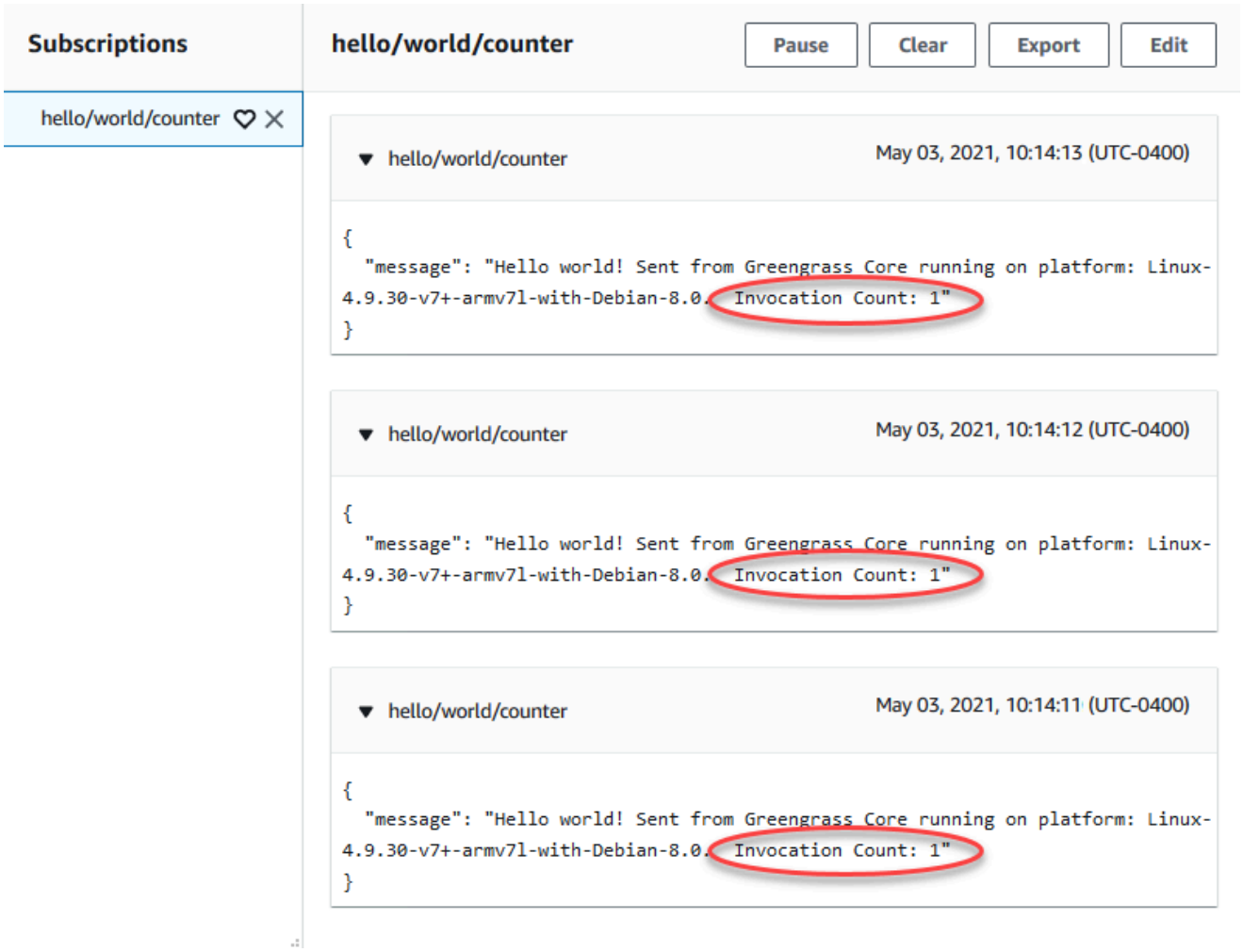
> **Note**
>
> No debería ver ningún mensaje después de suscribirse.

9. Para probar el ciclo de vida bajo demanda, invoque la función publicando un mensaje en el tema `hello/world/counter/trigger`. Puede utilizar el mensaje predeterminado.

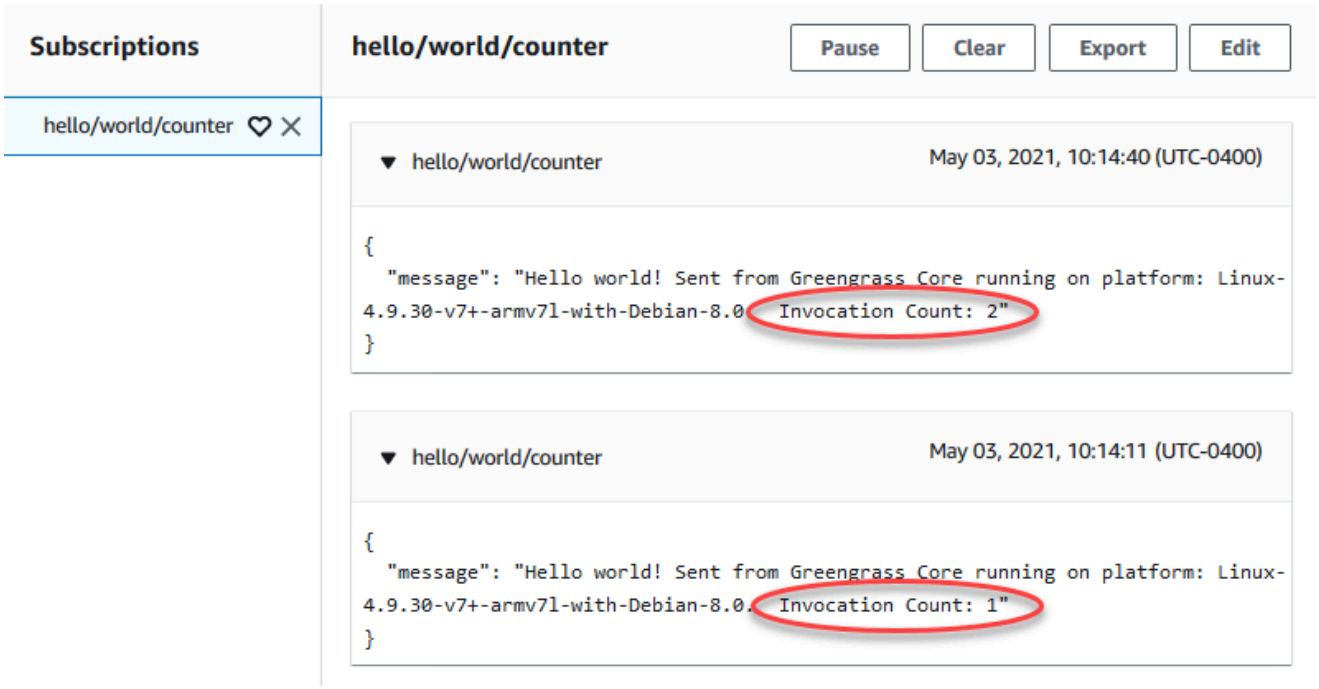
   a. Haga clic tres veces rápidamente en Publicar (cada pulsación del botón no debe sobrepasar los cinco segundos).



   Cada publicación invoca al controlador de la función y crea un contenedor para cada invocación. El número de invocaciones no aumenta durante las tres veces que se activa la función, ya que cada función de Lambda bajo demanda tiene su propio contenedor o entorno de pruebas.



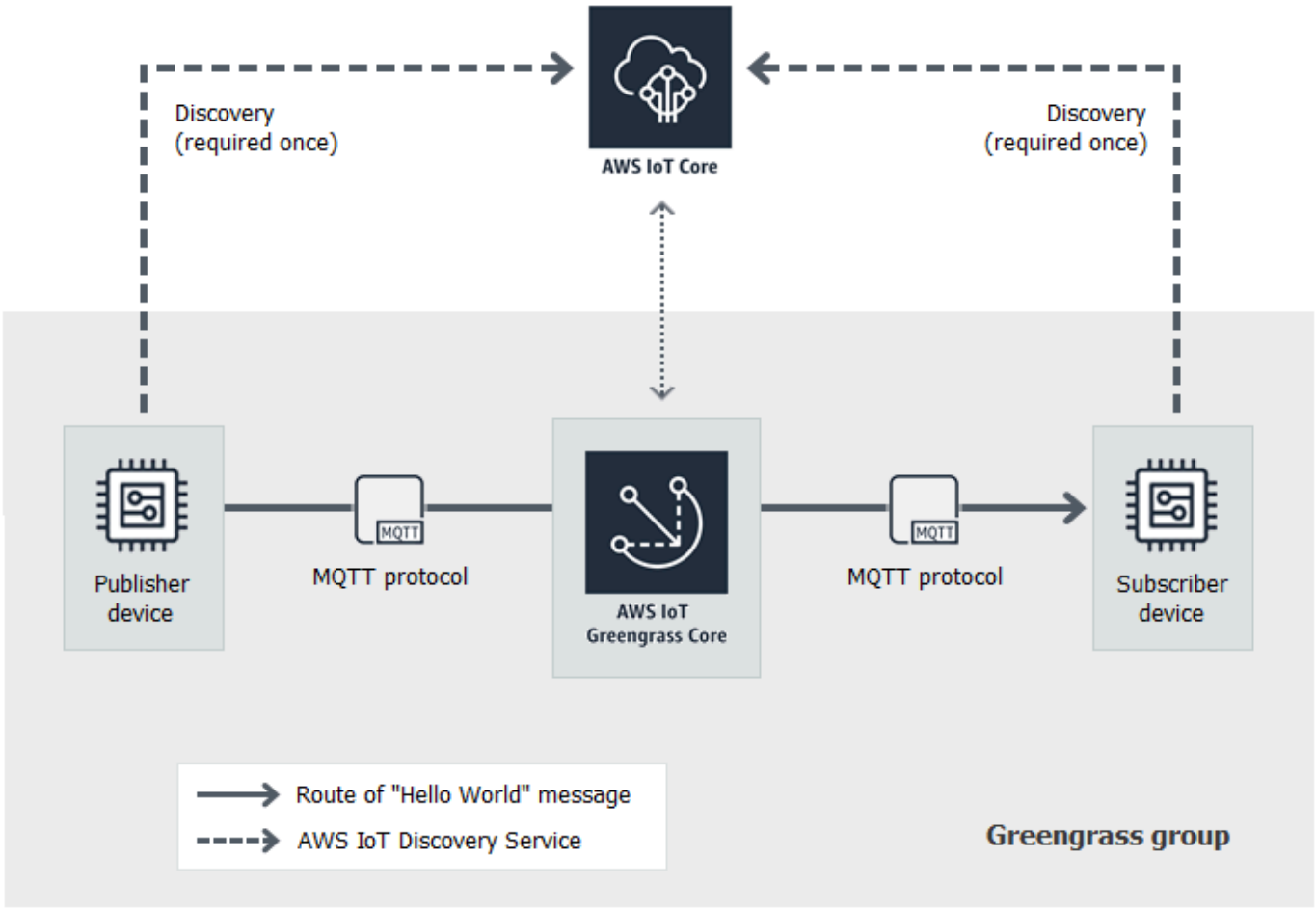
b. Después de aproximadamente 30 segundos, seleccione Publish to topic (Publicar en tema). El recuento de invocaciones debería incrementarse en 2. Esto demuestra que se está reutilizando un contenedor creado en una invocación anterior y que se han almacenado las variables de procesamiento previo externas al controlador de funciones.



Ahora debería entender los dos tipos de funciones de Lambda que se pueden ejecutar en el núcleo de AWS IoT Greengrass. El siguiente módulo, el Módulo 4, le muestra cómo pueden interactuar los dispositivos locales IoT en un grupo de AWS IoT Greengrass.

Módulo 4: Interacción con dispositivos cliente en un grupo de AWS IoT Greengrass

Este módulo muestra cómo los dispositivos IoT locales, denominados dispositivos del cliente o dispositivos, pueden conectarse y comunicarse con un dispositivo principal de AWS IoT Greengrass. Los dispositivos de l cliente que se conectan a un núcleo de AWS IoT Greengrass forman parte de un grupo de AWS IoT Greengrass y pueden participar en el paradigma de programación de AWS IoT Greengrass. En este módulo, un dispositivo de cliente envía un mensaje "Hello World" a otro dispositivo de cliente del grupo de Greengrass.



Antes de comenzar, ejecute el script de configuración del dispositivo de Greengrass o complete el módulo 1 y el módulo 2. En este módulo se crean dos dispositivos de cliente simulados. No necesita otros componentes ni dispositivos.

Completar este módulo debería tomarle menos de 30 minutos.

Temas

- Creación de dispositivos de en un grupo de AWS IoT Greengrass
- Configurar suscripciones
- Instale el SDK para dispositivos con AWS IoT para Python
- Probar las comunicaciones

Creación de dispositivos de en un grupo de AWS IoT Greengrass

En este paso, va a agregar dos dispositivos de cliente al grupo de Greengrass. Este proceso incluye el registro de los dispositivos como objetos de AWS IoT y la configuración de certificados y claves para permitir la conexión con AWS IoT Greengrass..

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. En la página de configuración del grupo, seleccione Dispositivos cliente y, a continuación, seleccione Asociar.
4. En el modal Asociar un dispositivo cliente a este grupo, seleccione Crear AWS IoTobjeto nuevo .

   La página Crear objetos se abre en una pestaña nueva.
5. En la página Crea objetos, elija Crear un solo objeto, y luego seleccione Siguiente.
6. En la página Especificar propiedades del objeto, registre este dispositivo cliente como **`HelloWorld_Publisher`** y, a continuación, seleccione Siguiente.
7. En la página Configurar el certificado del dispositivo, seleccione Siguiente.
8. En la página Adjuntar políticas al certificado, realice uno de los siguientes procedimientos:
   - Seleccione una política existente que conceda los permisos que requieren los dispositivos clientes y, a continuación, seleccione Crear objeto.

     Se abre un modal en el que puede descargar los certificados y las claves que el dispositivo utiliza para conectarse al Nube de AWS y al núcleo.
   - Cree y adjunte una nueva política que conceda permisos al dispositivo cliente. Haga lo siguiente:

     a. Elija Create Policy (Crear política).

        La página Create policy (Crear política) se abre en una pestaña nueva.

     b. En la página Create policy (Crear política), haga lo siguiente:

        i. En Nombre de política, introduzca un nombre que describa la política, como **`GreengrassV1ClientDevicePolicy`**.

        ii. En la pestaña Declaraciones de política, en Documento de política, seleccione JSON.

iii. Ingrese el siguiente documento de política. Esta política permite que el dispositivo cliente descubra los núcleos de Greengrass y comunique todos los temas MQTT. Para obtener información acerca de cómo restringir el acceso a esta política, consulte Autenticación y autorización de dispositivos en AWS IoT Greengrass.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. Elija Create (Crear) para crear la política.

c. Vuelva a la pestaña del navegador con la página Adjuntar políticas al certificado abierta. Haga lo siguiente:

i. En la lista Políticas, seleccione la política que ha creado, como GreengrassV1ClientDevicePolicy.

Si no se puede ver la política, seleccione el botón de actualizar.

ii. Elija Crear objeto.

Se abre un modal en el que puede descargar los certificados y las claves que el dispositivo utiliza para conectarse al Nube de AWS y al núcleo.

9. En el modal Descargar certificados y claves, descargue los certificados del dispositivo.

> **Important**
>
> Descargue los recursos de seguridad antes de elegir Listo.

Haga lo siguiente:

a. Para el certificado del dispositivo, seleccione Descargar para descargar el certificado del dispositivo.

b. En Archivo de clave pública, seleccione Descargar para descargar la clave pública del certificado.

c. En Archivo de clave privada, seleccione Descargar para descargar el archivo de clave privada del certificado.

d. Revise la Autenticación de servidor en la Guía del desarrollador de AWS IoT y seleccione el certificado de CA raíz adecuado. Le recomendamos que utilice los puntos de conexión de Amazon Trust Services (ATS) y los certificados de CA raíz de ATS. En Certificados de CA raíz, seleccione Descargar para obtener un certificado de CA raíz.

e. Seleccione Done (Listo).

Tome nota del identificador del certificado que comparten los nombres de archivo del certificado y las claves del dispositivo. Lo necesitará más adelante.

10. Regrese a la pestaña del navegador con el modal Asociar un dispositivo de cliente a este grupo abierto. Haga lo siguiente:

a. Para el nombre del objeto AWS IoT, seleccione el objeto HelloWorld_Publisher que creó.

Si no se puede ver el objeto, seleccione el botón de actualizar.

b. Elija Associate (Asociar).

11. Repita los pasos 3 a 10 para añadir un segundo dispositivo cliente al grupo.

Asigne un nombre a este dispositivo cliente **`HelloWorld_Subscriber`**. Descargue los certificados y las claves del dispositivo del cliente en el equipo. De nuevo, tome nota del ID del certificado que es común en los nombres de archivo del dispositivo HelloWorld_Subscriber.

Ahora debería tener dos dispositivos cliente en su grupo de Greengrass:

- HelloWorld_Publisher
- HelloWorld_Subscriber

12. Cree una carpeta en su equipo para las credenciales de seguridad de estos dispositivos cliente. Copie los certificados y las claves en esta carpeta.

Configurar suscripciones

En este paso, va a permitir que el dispositivo de cliente HelloWorld_Publisher envíe mensajes MQTT al dispositivo de cliente HelloWorld_Subscriber.

1. En la página de configuración de grupo, elija la pestaña Suscripciones y, a continuación, elija Agregar.
2. En la página Crear suscripción haga lo siguiente para configurar la suscripción:

   a. En Tipo de origen, elija Dispositivo de cliente y, a continuación, HelloWorld_Publisher.

   b. En Tipo de destino, elija Dispositivo de cliente y, a continuación, seleccione HelloWorld_Subscriber.

   c. En Filtro de temas, escriba **`hello/world/pubsub`**.

   > **Note**
   >
   > Puede eliminar las suscripciones de los módulos anteriores. En la página Suscripciones del grupo, seleccione las suscripciones que desea eliminar y, a continuación, elija Eliminar.

   d. Elija Crear una suscripción.
3. Asegúrese de que la detección automática está habilitada para que el núcleo de Greengrass pueda publicar una lista de sus direcciones IP. Los dispositivos cliente usan esta información para descubrir el núcleo. Haga lo siguiente:

   a. En la página de configuración del grupo, elija la pestaña Funciones de lambda.

   b. En Funciones de Lambda del sistema, elija Detector IP y, a continuación, seleccione Editar.
   c. En la Configuración Editar detector IP, seleccione Detectar y anular automáticamente los puntos de conexión del agente MQTT y, a continuación, seleccione Guardar.
4. Asegúrese de que el daemon de Greengrass esté en ejecución, tal y como se describe en Implementación de configuraciones de nube en un dispositivo central.
5. En la página de configuración de grupo, elija Implementar.

El estado de implementación aparece en la parte superior de la página, bajo el nombre del grupo. Para ver los detalles de la implementación, seleccione la pestaña Implementaciones.

Instale el SDK para dispositivos con AWS IoT para Python

Los dispositivos cliente pueden utilizar el SDK para dispositivos con AWS IoT para Python para comunicarse con los dispositivos principales AWS IoT y AWS IoT Greengrass (utilizando el lenguaje de programación Python). Para obtener más información, incluidos los requisitos, consulte el SDK para dispositivos con AWS IoT para ver el Readme de Python en GitHub.

En este paso, instalará el SDK y obtendrá la función de ejemplo `basicDiscovery.py`, que utilizan los dispositivos de cliente simulados en el equipo.

1. Para instalar el SDK con todos los componentes necesarios en el equipo, elija el sistema operativo:

   Windows

   1. Abra un símbolo del sistema elevado y ejecute el siguiente comando:

      ```
      python --version
      ```

      Si no devuelve información sobre la versión o si la versión es inferior a 2.7 para Python 2 o a 3.3 para Python 3, siga las instrucciones de la página Downloading Python (Descargar Python) para instalar Python 2.7+ o Python 3.3+. Para obtener más información acerca de los periodos de mantenimiento, consulte Using Python on Windows.
   2. Descargue SDK para dispositivos con AWS IoT para Python como un archivo `zip` y extráigalo en la ubicación adecuada del equipo.

Anote la ruta del archivo en la carpeta `aws-iot-device-sdk-python-master` extraída que contiene el archivo `setup.py`. En el siguiente paso, *`ruta-a-carpeta-SDK`* indica esta ruta.

3. Ejecute el siguiente comando desde el símbolo del sistema elevado:

```
cd path-to-SDK-folder
python setup.py install
```

macOS

1. Abra una ventana de terminal y ejecute el siguiente comando:

```
python --version
```

Si no devuelve información sobre la versión o si la versión es inferior a 2.7 para Python 2 o a 3.3 para Python 3, siga las instrucciones de la página Downloading Python (Descargar Python) para instalar Python 2.7+ o Python 3.3+. Para obtener más información acerca de los periodos de mantenimiento, consulte Using Python on a Macintosh.

2. En la ventana de terminal, ejecute los siguientes comandos para determinar la versión de OpenSSL:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Anote el valor de versión de OpenSSL.

> **Note**
>
> Si está ejecutando Python 3, use print(ssl.OPENSSL_VERSION).

Para cerrar manualmente el shell de Python, ejecute el siguiente comando:

```
>>>exit()
```

Si la versión de OpenSSL es 1.0.1 o posterior, vaya al paso c. De lo contrario, siga estos pasos:

- En la ventana de terminal, ejecute el siguiente comando para determinar si el equipo está utilizando Simple Python Version Management:

```
which pyenv
```

Si devuelve una ruta de archivo, elija la pestaña Con **pyenv**. Si no devuelve nada, elija la pestaña Sin **pyenv**.

Using pyenv

1. Consulte Python Releases for Mac OS X (o similar) para informarse acerca de la última versión de Python estable. En el ejemplo siguiente, este valor se indica mediante *última-versión-de-Python*.
2. En una ventana de terminal, ejecute los siguientes comandos:

   ```
   pyenv install latest-Python-version
   pyenv global latest-Python-version
   ```

   Por ejemplo, si la última versión de Python 2 es 2.7.14, estos comandos son:

   ```
   pyenv install 2.7.14
   pyenv global 2.7.14
   ```

3. Cierre y, a continuación, vuelva a abrir la ventana del terminal y, a continuación, ejecute los comandos siguientes:

   ```
   python
   >>>import ssl
   >>>print ssl.OPENSSL_VERSION
   ```

   La versión de OpenSSL debe ser 1.0.1 o posterior. Si la versión es inferior a 1.0.1, entonces la actualización ha fallado. Compruebe el valor de la versión de Python utilizado en los comandos pyenv global y pyenv install e inténtelo de nuevo.

4. Ejecute el siguiente comando para cerrar el shell de Python:

```
exit()
```

Not using pyenv

1. Desde una ventana de terminal, ejecute el siguiente comando para determinar si brew está instalado.

   ```
   which brew
   ```

   Si no devuelve una ruta, instale `brew` de la siguiente manera:

   ```
   /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
   Homebrew/install/master/install)"
   ```

   > **Note**
   >
   > Siga las instrucciones de instalación. La descarga de las herramientas de línea de comandos de Xcode pueden tardar algún tiempo.

2. Ejecute los comandos siguientes:

   ```
   brew update
   brew install openssl
   brew install python@2
   ```

   El SDK para dispositivos con AWS IoT para Python requiere OpenSSL versión 1.0.1 (o posterior) compilado con el ejecutable de Python. El comando brew install python instala un ejecutable `python2` que cumple este requisito. El ejecutable `python2` se instala en el directorio `/usr/local/bin`, que debe ser parte de la variable de entorno `PATH`. Para confirmar, ejecute el siguiente comando:

   ```
   python2 --version
   ```

   Si se proporciona información de la versión de `python2`, vaya al siguiente paso. De lo contrario, agregue la siguiente línea a su perfil de shell para añadir de forma permanente la ruta `/usr/local/bin` a la variable de entorno `PATH`:

```
export PATH="/usr/local/bin:$PATH"
```

Por ejemplo, si está utilizando `.bash_profile` o aún no dispone de un perfil de shell, ejecute este comando desde una ventana de terminal:

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

A continuación, ejecute el comando source en el perfil de shell y compruebe si `python2 --version` proporciona información de la versión. Por ejemplo, si está utilizando `.bash_profile`, ejecute los siguientes comandos.

```
source ~/.bash_profile
python2 --version
```

Debe devolver información de la versión de `python2`.

3. Agregue la siguiente línea a su perfil de shell:

```
alias python="python2"
```

Por ejemplo, si está utilizando `.bash_profile` o aún no dispone de un perfil de shell, ejecute este comando:

```
echo 'alias python="python2"' >> ~/.bash_profile
```

4. A continuación, ejecute el comando source en el perfil de shell. Por ejemplo, si está utilizando `.bash_profile`, ejecute el siguiente comando:

```
source ~/.bash_profile
```

Al invocar el comando python se ejecuta el ejecutable de Python que contiene la versión de OpenSSL necesaria (`python2`).

5. Ejecute los comandos siguientes:

```
python
 import ssl
 print ssl.OPENSSL_VERSION
```

La versión de OpenSSL debe ser 1.0.1 o posterior.

6. Para cerrar el shell de Python, ejecute siguiente comando:

```
exit()
```

3. Ejecute los siguientes comandos para instalar el SDK para dispositivos con AWS IoT para Python:

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

UNIX-like system

1. En una ventana de terminal de , ejecute el siguiente comando:

```
python --version
```

Si no devuelve información sobre la versión o si la versión es inferior a 2.7 para Python 2 o a 3.3 para Python 3, siga las instrucciones de la página Downloading Python (Descargar Python) para instalar Python 2.7+ o Python 3.3+. Para obtener más información acerca de los periodos de mantenimiento, consulte Using Python on Unix platforms.

2. En el terminal, ejecute los siguientes comandos para determinar la versión de OpenSSL:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Anote el valor de versión de OpenSSL.

> **Note**
>
> Si está ejecutando Python 3, use print(ssl.OPENSSL_VERSION).

Para cerrar manualmente el shell de Python, ejecute el siguiente comando:

```
exit()
```

Si la versión de OpenSSL es 1.0.1 o posterior, vaya al paso siguiente. De lo contrario, ejecute los comandos necesarios para actualizar OpenSSL según su distribución (por ejemplo, `sudo yum update openssl`, `sudo apt-get update`, etc.).

Ejecute los siguientes comandos para determinar si la versión de OpenSSL es 1.0.1 o posterior:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
>>>exit()
```

3. Ejecute los siguientes comandos para instalar el SDK para dispositivos con AWS IoT para Python:

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

2. Una vez que el SDK para dispositivos con AWS IoT para Python esté instalado, vaya a la carpeta `samples` y abra la carpeta `greengrass`.

   Para este tutorial, copiará la función `basicDiscovery.py` de ejemplo, que utiliza los certificados y las claves que ha descargado en the section called “Creación de dispositivos de en un grupo de AWS IoT Greengrass”.

3. Copie `basicDiscovery.py` en la carpeta que contiene las claves y certificados de dispositivo HelloWorld_Publisher y HelloWorld_Subscriber.

Probar las comunicaciones

1. Asegúrese de que el ordenador y el dispositivo AWS IoT Greengrass principal estén conectados a Internet mediante la misma red.

a. En el dispositivo AWS IoT Greengrass principal, ejecuta el siguiente comando para buscar su dirección IP.

```
hostname -I
```

b. En el equipo, ejecute el siguiente comando utilizando la dirección IP del dispositivo central. Puede utilizar Ctrl + C para detener el comando ping.

```
ping IP-address
```

Un resultado similar al siguiente indica que la comunicación entre el ordenador y el dispositivo AWS IoT Greengrass principal se ha realizado correctamente (pérdida de paquetes del 0%):

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

> **Note**
>
> Si no puede hacer ping a una instancia de EC2 en ejecución AWS IoT Greengrass, asegúrese de que las reglas del grupo de seguridad entrante de la instancia permitan el tráfico ICMP para los mensajes de solicitud de Echo. Para obtener más información, consulte Añadir reglas a un grupo de seguridad en la Guía del usuario de Amazon EC2.
>
> Es posible que, en los equipos host de Windows, en la aplicación Firewall de Windows con seguridad avanzada, también tenga que habilitar una regla de entrada que permita las solicitudes de eco entrantes (por ejemplo, Compartir archivos e impresoras [Solicitud de eco - ICMPv4-In]) o crear una.

2. Obtenga su AWS IoT punto final.

   a. En el panel de navegación de la consola de AWS IoT, seleccione Configuración.

   b. En Punto de conexión de datos del dispositivo, anote el valor del punto de conexión. Este valor se usa para sustituir el marcador de posición *AWS_IOT_ENDPOINT* de los comandos en los pasos que se describen a continuación.

   > **Note**
   >
   > Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.

3. En su ordenador (no en el dispositivo AWS IoT Greengrass principal), abra dos ventanas de línea de comandos (terminal o línea de comandos). Una ventana representa el dispositivo cliente HelloWorld _Publisher y la otra representa el dispositivo cliente HelloWorld _Subscriber.

   Tras la ejecución, `basicDiscovery.py` intenta recopilar información sobre la ubicación del AWS IoT Greengrass núcleo en sus puntos finales. Esta información se almacena una vez que el dispositivo del cliente ha detectado y establecido la conexión correctamente con el dispositivo central. De este modo, la mensajería y las operaciones que se realicen en el futuro se ejecutarán localmente (sin necesidad de tener conexión a Internet).

   > **Note**
   >
   > Los ID de cliente utilizados para las conexiones MQTT deben coincidir con el nombre del dispositivo de cliente. El script `basicDiscovery.py` establece el ID del cliente para las conexiones MQTT con el nombre del objeto que especifique al ejecutar el script.
   > Ejecute el siguiente comando desde la carpeta que contiene el archivo `basicDiscovery.py` de la información de uso detallada del script:
   >
   > ```
   > python basicDiscovery.py --help
   > ```

4. Desde la ventana del dispositivo cliente HelloWorld _Publisher, ejecute los siguientes comandos.

   - Sustituya *path-to-certs-folder* por la ruta de la carpeta que contiene los certificados, las claves y `basicDiscovery.py`.
   - Sustituya *AWS_IOT_ENDPOINT* por el punto de enlace.

- Sustituya las dos CertId instancias del *editor* por el identificador del certificado en el nombre de archivo de su dispositivo cliente HelloWorld _Publisher.

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
 --cert publisherCertId-certificate.pem.crt --key publisherCertId-private.pem.key
 --thingName HelloWorld_Publisher --topic 'hello/world/pubsub' --mode publish --
message 'Hello, World! Sent from HelloWorld_Publisher'
```

El resultado debería ser similar al siguiente, que contiene entradas como `Published topic 'hello/world/pubsub': {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}`.

> **Note**
>
> Si el script devuelve un mensaje `error: unrecognized arguments`, cambie las comillas simples por comillas dobles en los parámetros `--message` y `--topic` y vuelva a ejecutar el comando.
> Para resolver un problema de conexión, puede intentar usar la detección manual de IP.

```
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}

2017-11-13 21:12:26,296 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:26,297 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:27,301 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}

2017-11-13 21:12:27,302 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:27,303 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:28,305 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}

2017-11-13 21:12:28,306 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:28,307 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:29,310 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 3}
```

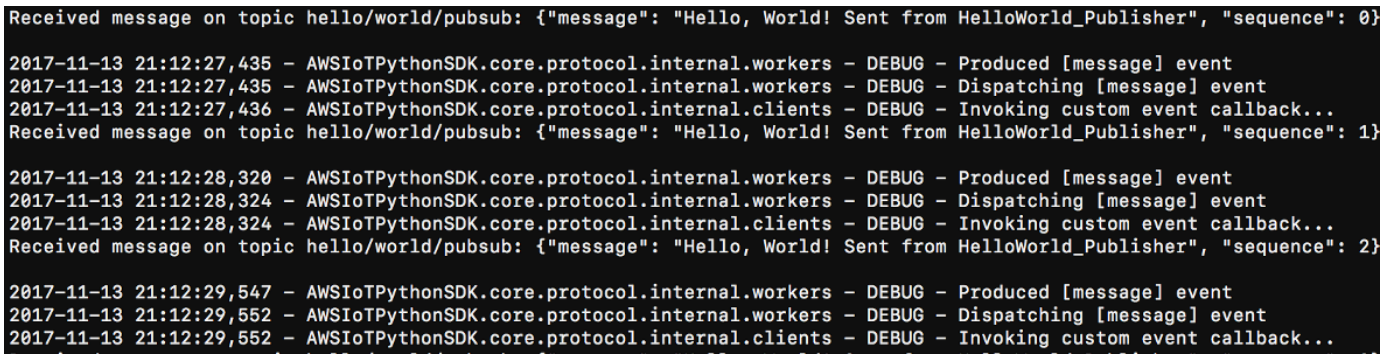
5. En la ventana del dispositivo cliente HelloWorld _Subscriber, ejecute los siguientes comandos.

   - Sustituya *path-to-certs-folder* por la ruta de la carpeta que contiene los certificados, las claves y `basicDiscovery.py`.
   - Sustituya *AWS_IOT_ENDPOINT* por el punto de enlace.

- Sustituya las dos CertId instancias de *suscriptor* por el ID del certificado en el nombre de archivo de su dispositivo cliente HelloWorld _Subscriber.

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
cert subscriberCertId-certificate.pem.crt --key subscriberCertId-private.pem.key --
thingName HelloWorld_Subscriber --topic 'hello/world/pubsub' --mode subscribe
```

El resultado debería ser similar al siguiente, que contiene entradas como `Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}`.



Cierre la ventana HelloWorld_Publisher para evitar que los mensajes se acumulen en la ventana HelloWorld_Subscriber.

Si se realizan pruebas en una red corporativa, la conexión con el dispositivo central podría verse afectada. Para evitarlo, puede especificar manualmente el punto de enlace. Esto garantiza que el `basicDiscovery.py` script se conecte a la dirección IP correcta del dispositivo AWS IoT Greengrass principal.

Para especificar manualmente el punto de enlace

1. En el panel de navegación de la AWS IoT consola, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. En Grupos de Greengrass, elija su grupo.
3. Configure el núcleo para gestionar manualmente los puntos de conexión del agente de MQTT. Haga lo siguiente:

   a. En la página de configuración del grupo, elija la pestaña Funciones de Lambda.

b. En Funciones de Lambda del sistema, elija Detector IP y, a continuación, seleccione Editar.

c. En la Editar la configuración del detector IP, seleccione Administrar manualmente los puntos de conexión del agente MQTT y, a continuación, seleccione Guardar.

4. Introduzca el punto de conexión del agente de MQTT para el núcleo. Haga lo siguiente:

a. En Información general, seleccione Núcleo de Greengrass.

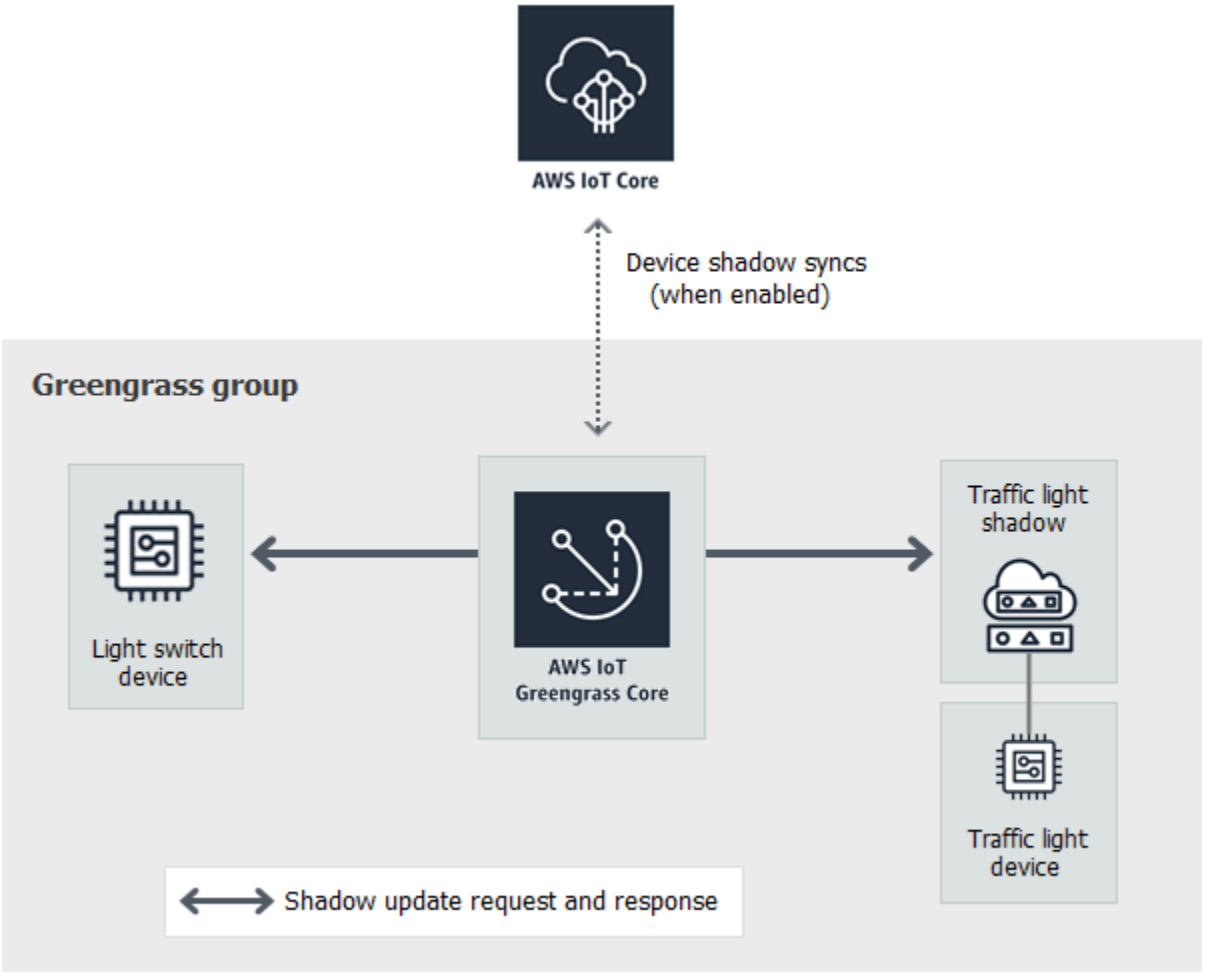
b. En Puntos de conexión del agente de MQTT, seleccione Administrar puntos de conexión.

c. Seleccione Añadir punto de conexión y asegúrese de que solo tiene un valor de punto de conexión. Este valor debe ser el punto final de la dirección IP del puerto 8883 del dispositivo AWS IoT Greengrass principal (por ejemplo,`192.168.1.4`).

d. Elija Actualizar.

Módulo 5: Interacción con sombras de dispositivos

En este módulo avanzado, se muestra cómo pueden interactuar los dispositivos del cliente con sombras de dispositivos de AWS IoT en un grupo de AWS IoT Greengrass. Una sombra es un documento JSON que se usa para almacenar la información del estado actual o deseado de un objeto. En este módulo, descubrirá cómo un dispositivo de cliente (`GG_Switch`) puede modificar el estado de otro dispositivo de cliente (`GG_TrafficLight`) y cómo estos estados pueden sincronizarse con la nube de AWS IoT Greengrass:



Antes de comenzar, ejecute el script de configuración de dispositivos de Greengrass o asegúrese de haber completado el módulo 1 y el módulo 2. También debería comprender el procedimiento para conectar dispositivos a un núcleo de AWS IoT Greengrass (Módulo 4). No necesita otros componentes ni dispositivos.

Completar este módulo debería tomarle aproximadamente 30 minutos.

Temas

- Configurar dispositivos y suscripciones
- Descargar los archivos necesarios
- Probar las comunicaciones (sincronizaciones de dispositivos deshabilitadas)
- Probar las comunicaciones (sincronizaciones de dispositivos habilitadas)

Configurar dispositivos y suscripciones

Las sombras pueden sincronizarse en AWS IoT cuando AWS IoT Greengrass core se conecta a Internet. Este módulo, utilizará primero las sombras locales sin sincronizar con la nube. Después, habilitará la sincronización con la nube.

Cada dispositivo cliente tiene su propia sombra. Para obtener más información, consulte el servicio sombra de dispositivo para AWS IoT en la Guía del desarrollador deAWS IoT.

1. En la página de configuración de grupo, elija Dispositivos cliente.
2. En la pestaña Dispositivos cliente, añada dos nuevos dispositivos cliente a su grupo AWS IoT Greengrass. Para obtener información detallada sobre este proceso, consulte the section called “Creación de dispositivos de en un grupo de AWS IoT Greengrass”.
   - Utilice los nombres **`GG_Switch`** y **`GG_TrafficLight`** para los dispositivos cliente.
   - Genere y descargue los recursos de seguridad para ambos dispositivos cliente.
   - Anote el ID del certificado en los nombres de archivo de los recursos de seguridad para los dispositivos cliente. Utilizará estos nombres más adelante.
3. Cree una carpeta en su ordenador para las credenciales de seguridad de estos dispositivos cliente. Copie los certificados y las claves en esta carpeta.
4. Asegúrese de que los dispositivos cliente están configurados para utilizar sombras locales y no sincronizarse con la Nube de AWS. Si no es así, seleccione el dispositivo cliente, seleccione Sincronizar sombra y, a continuación, seleccione Desactivar sincronización oculta con la nube.
5. Añada las suscripciones de la tabla siguiente al grupo. Por ejemplo, para crear la primera suscripción:
   a. En la página de configuración de grupo, elija la pestaña Suscripciones y, a continuación, elija Agregar.
   b. En Tipo de origen, elija Dispositivo cliente y, a continuación, elija GG_Switch.
   c. Para Seleccionar un destino: elija Servicios y, a continuación, elija Servicio de sombra local.
   d. En Filtro de temas, escriba **`$aws/things/GG_TrafficLight/shadow/update`**.
   e. Elija Crear una suscripción.

   Los temas deben escribirse exactamente igual que aparecen en la tabla. Aunque puede utilizar caracteres comodín para consolidar algunas de las suscripciones, no le recomendamos que

lo haga. Para obtener más información, consulte Temas MQTT de sombra en la Guía para desarrolladores de AWS IoT.

Origen	Objetivo	Tema	Notas
GG_Switch	Servicio de sombra local	$aws/things/GG_TrafficLight/shadow/update	GG_Switch envía una solicitud de actualización al tema de actualización.
Servicio de sombra local	GG_Switch	$aws/things/GG_TrafficLight/shadow/update/accepted	GG_Switch necesita saber si se aceptó la solicitud de actualización.
Servicio de sombra local	GG_Switch	$aws/things/GG_TrafficLight/shadow/update/rejected	GG_Switch necesita saber si se rechazó la solicitud de actualización.
GG_TrafficLight	Servicio de sombra local	$aws/things/GG_TrafficLight/shadow/update	GG_TrafficLight envía una actualización de su estado al tema de actualización.
Servicio de sombra local	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/delta	El servicio de sombra local envía una actualización de recepción a GG_TrafficLight a través del tema delta.
Servicio de sombra local	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/accepted	GG_TrafficLight necesita saber si se aceptó la actualización de estado.

Origen	Objetivo	Tema	Notas
Servicio de sombra local	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/rejected	GG_TrafficLight necesita saber si se rechazó la actualización de estado.

Las suscripciones nuevas se muestran en la pestaña Suscripciones.

> **Note**
>
> Para obtener más información sobre el carácter $, consulte Temas reservados.

6. Asegúrese de que la detección automática está habilitada para que el núcleo de Greengrass pueda publicar una lista de sus direcciones IP. Los dispositivos cliente usan esta información para descubrir el núcleo. Haga lo siguiente:

   a. En la página de configuración del grupo, elija la pestaña Funciones de lambda.
   b. En Funciones de Lambda del sistema, elija Detector IP y, a continuación, seleccione Editar.
   c. En la Configuración Editar detector IP, seleccione Detectar y anular automáticamente los puntos de conexión del agente MQTT y, a continuación, seleccione Guardar.

7. Asegúrese de que el daemon de Greengrass esté en ejecución, tal y como se describe en Implementación de configuraciones de nube en un dispositivo central.
8. En la página de configuración de grupo, elija Implementar.

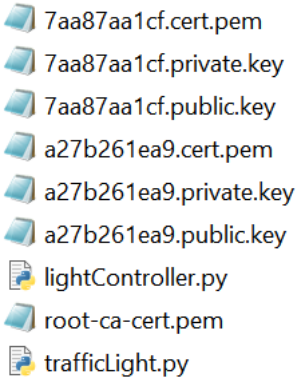
Descargar los archivos necesarios

1. Si aún no lo ha hecho, instale el SDK para dispositivos con AWS IoT para Python. Para obtener instrucciones, consulte el paso 1 en the section called “Instale el SDK para dispositivos con AWS IoT para Python”.

   Este SDK lo utilizan todos los dispositivos de cliente para comunicarse con AWS IoT y con los dispositivos de AWS IoT Greengrass Core.

2. Desde la carpeta de ejemplos TrafficLight en GitHub, descargue los archivos `trafficLight.py` y `lightController.py` en su equipo. Guárdelos en la carpeta que contiene los certificados y las claves de los dispositivos de cliente GG_TrafficLight y GG_Switch.

El script `lightController.py` se corresponde con el dispositivo de cliente GG_Switch y el script `trafficLight.py` con el dispositivo de cliente GG_TrafficLight.



> **Note**
>
> Los archivos Python de ejemplo se almacenan en el repositorio del SDK AWS IoT Greengrass Core para Python por comodidad, pero no utilizan el SDK AWS IoT Greengrass Core.

Probar las comunicaciones (sincronizaciones de dispositivos deshabilitadas)

1. Asegúrese de que el ordenador y el dispositivo AWS IoT Greengrass principal estén conectados a Internet mediante la misma red.

   a. En el dispositivo AWS IoT Greengrass principal, ejecuta el siguiente comando para buscar su dirección IP.

   ```
   hostname -I
   ```

   b. En el equipo, ejecute el siguiente comando utilizando la dirección IP del dispositivo central. Puede utilizar Ctrl + C para detener el comando ping.

   ```
   ping IP-address
   ```

Un resultado similar al siguiente indica que la comunicación entre el ordenador y el dispositivo AWS IoT Greengrass principal se ha realizado correctamente (pérdida de paquetes del 0%):

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

> **Note**
>
> Si no puede hacer ping a una instancia de EC2 en ejecución AWS IoT Greengrass, asegúrese de que las reglas del grupo de seguridad entrante de la instancia permitan el tráfico ICMP para los mensajes de solicitud de Echo. Para obtener más información, consulte Añadir reglas a un grupo de seguridad en la Guía del usuario de Amazon EC2.
>
> Es posible que, en los equipos host de Windows, en la aplicación Firewall de Windows con seguridad avanzada, también tenga que habilitar una regla de entrada que permita las solicitudes de eco entrantes (por ejemplo, Compartir archivos e impresoras [Solicitud de eco - ICMPv4-In]) o crear una.

2. Obtenga su AWS IoT punto final.

   a. En el panel de navegación de la consola de AWS IoT, seleccione Configuración.

   b. En Punto de conexión de datos del dispositivo, anote el valor del punto de conexión. Este valor se usa para sustituir el marcador de posición *AWS_IOT_ENDPOINT* de los comandos en los pasos que se describen a continuación.

> **Note**
>
> Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.

3. En su ordenador (no en el dispositivo AWS IoT Greengrass principal), abra dos ventanas de línea de comandos (terminal o línea de comandos). Una ventana representa el dispositivo cliente GG_Switch y la otra representa el dispositivo cliente GG_. TrafficLight

   a. En la ventana del dispositivo de cliente GG_Switch, ejecute los comandos siguientes.

   - Sustituya *path a certs-folder* por la ruta de la carpeta que contiene los certificados, las claves y los archivos de Python.
   - Sustituya *AWS_IOT_ENDPOINT* por el punto de enlace.
   - Sustituya las dos CertId instancias del *conmutador* por el ID del certificado en el nombre de archivo de su dispositivo cliente GG_Switch.

```
cd path-to-certs-folder
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA
 AmazonRootCA1.pem --cert switchCertId-certificate.pem.crt --key switchCertId-
private.pem.key --thingName GG_TrafficLight --clientId GG_Switch
```

   b. Desde la ventana del dispositivo TrafficLight cliente GG_, ejecute los siguientes comandos.

   - Sustituya *path a certs-folder* por la ruta de la carpeta que contiene los certificados, las claves y los archivos de Python.
   - Sustituya *AWS_IOT_ENDPOINT* por el punto de enlace.
   - Sustituya las dos CertId instancias *ligeras* por el ID del certificado en el nombre de archivo de su dispositivo TrafficLight cliente GG_.

```
cd path-to-certs-folder
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
 --cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --
thingName GG_TrafficLight --clientId GG_TrafficLight
```

Cada 20 segundos, el interruptor actualiza el estado de sombra a los valores G, Y y R, y la luz mostrará su nuevo estado, tal que se muestra a continuación.

Salida de GG_Switch:

```
{"state":{"desired":{"property":"R"}}}
2018-12-20 12:23:01,446 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~~~~~~~Shadow Update Accepted~~~~~~~~~~~~~~~
Update request with token: 3b22e27c-930d-4c6a-8562-9f86088249f4 accepted!
property: R
~~~~~~~~~~~~~~~~~~~~~~~~~
```

Salida GG_TrafficLight :

```
++++++++ Received Shadow Delta ++++++++++
{u'state': {u'property': u'R'}, u'metadata': {u'property': {u'timestamp': 1545337381}}, u'version': 33, u'clientToken':
u'3b22e27c-930d-4c6a-8562-9f86088249f4'}
property: R
version: 33
++++++++++++++++++++++


Light changed to: R
{"state":{"reported":{"property":"R"}}}
2018-12-20 12:23:01,539 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~~~~~~ Shadow Update Accepted ~~~~~~~~~~~~~~
Update request with token: f552109f-c1c2-4ae6-a841-8443506eefcb accepted!
property: R
~~~~~~~~~~~~~~~~~~~~~~~~~
```

Cuando se ejecuta por primera vez, cada script del dispositivo cliente ejecuta el servicio de AWS IoT Greengrass detección para conectarse al AWS IoT Greengrass núcleo (a través de Internet). Una vez que un dispositivo cliente haya descubierto y se haya conectado correctamente al AWS IoT Greengrass núcleo, las operaciones futuras se pueden ejecutar localmente.

> **Note**
>
> Los scripts `lightController.py` y `trafficLight.py` almacenan información de conexión en la carpeta `groupCA`, que se crea en la misma carpeta que los scripts. Si recibe errores de conexión, asegúrese de que la dirección IP del archivo `ggc-host` coincide con el punto de conexión de la dirección IP de su núcleo.

4. En la AWS IoT consola, elija su AWS IoT Greengrass grupo, elija la pestaña Dispositivos cliente y, a continuación, elija GG_ TrafficLight para abrir la página de detalles del AWS IoT dispositivo cliente.

5. Seleccione la pestaña Sombras de dispositivo. Después de que el GG_Switch cambie de estado, no debería haber ninguna actualización de esta sombra. Esto se debe a que el GG_ TrafficLight está configurado para deshabilitar la sincronización oculta con la nube.
6. Pulse Ctrl + C en la ventana del dispositivo de cliente GG_Switch (`lightController.py`). Deberías ver que la ventana GG_ TrafficLight (`trafficLight.py`) deja de recibir mensajes de cambio de estado.

   Mantenga estas ventanas abiertas para que pueda ejecutar los comandos de la siguiente sección.

Probar las comunicaciones (sincronizaciones de dispositivos habilitadas)

En esta prueba, va a configurar la sombra del dispositivo GG_TrafficLight para que se sincronice con AWS IoT. Para ello, tendrá que ejecutar los mismos comandos que en la prueba anterior, pero esta vez el estado de la sombra en la nube se actualizará cuando GG_Switch envíe una solicitud de actualización.

1. En la consola de AWS IoT, elija su grupo AWS IoT Greengrass y, a continuación, elija la pestaña Dispositivos cliente.
2. Seleccione el dispositivo GG_TrafficLight, elija Sincronizar sombra y, a continuación, seleccione Activar sincronización de sombra con la nube.

   Debería recibir una notificación de que se ha actualizado el estado de sincronización de la sombra de dispositivo.
3. En la página de configuración de grupo, elija Implementar.
4. En las dos ventanas de línea de comandos, ejecute los comandos de la prueba anterior para los dispositivos cliente GG_Switch y GG_TrafficLight.
5. Ahora, compruebe el estado de la sombra en la consola de AWS IoT. Elija su grupo AWS IoT Greengrass, elija la pestaña Dispositivos cliente, elija GG_TrafficLight, elija la pestaña Sombras de dispositivos y, a continuación, elija Sombra clásica.

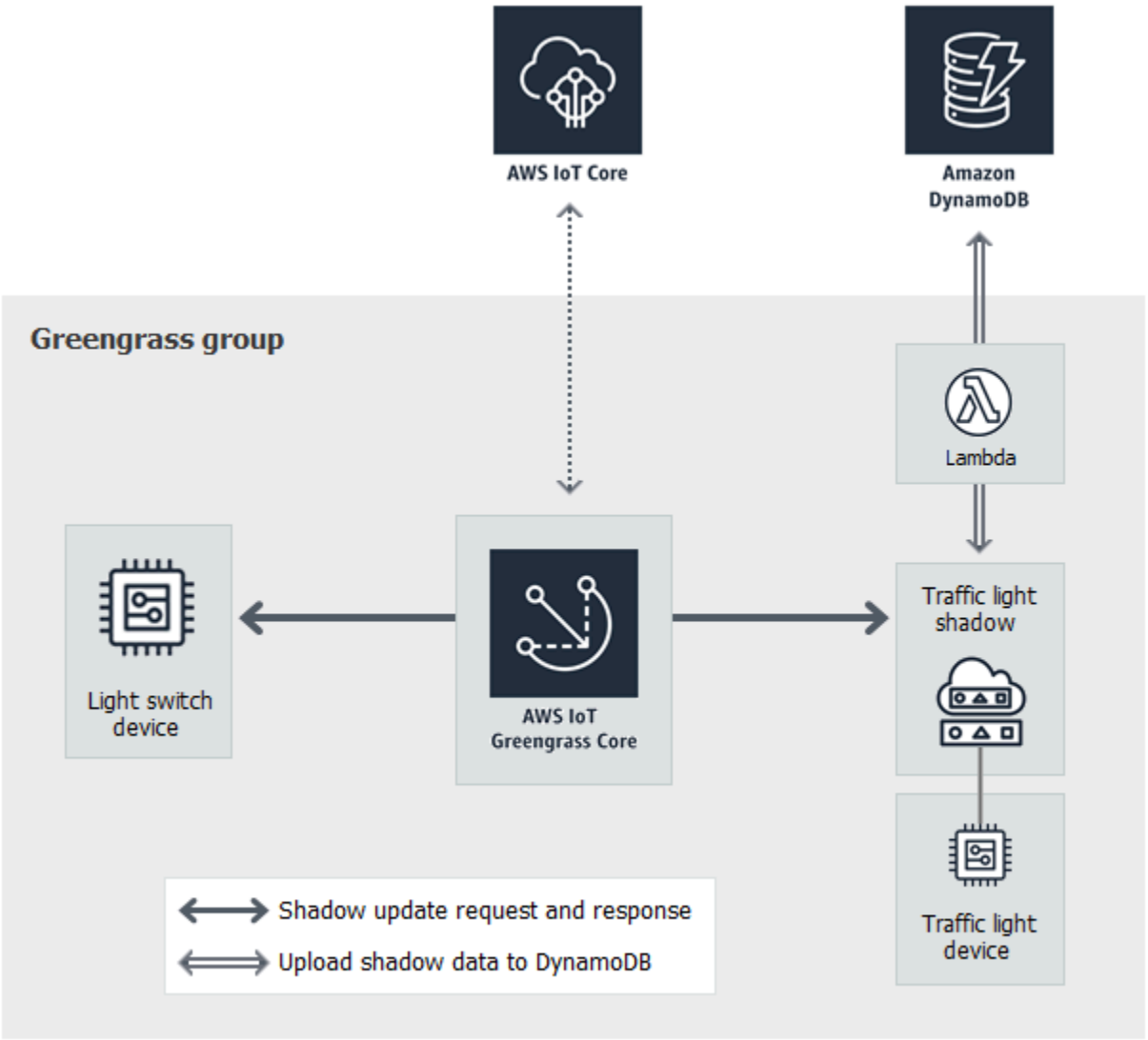
   Como habilitó la sincronización de la sombra de GG_TrafficLight con AWS IoT, el estado de la sombra en la nube debería actualizarse cuando GG_Switch envíe una actualización. Esta funcionalidad puede usarse para exponer el estado de un dispositivo cliente en AWS IoT.

> **Note**
>
> Para solucionar algún problema que pueda surgir, puede consultar los registros de AWS IoT Greengrass del núcelo, en particular `runtime.log`:
>
> ```
> cd /greengrass/ggc/var/log
> sudo cat system/runtime.log | more
> ```
>
> También puede ver `GGShadowSyncManager.log` y `GGShadowService.log`. Para obtener más información, consulte Solución de problemas.

Mantenga los dispositivos cliente y las suscripciones configurados. Los usará en el módulo siguiente. También puede ejecutar los mismos comandos.

Módulo 6: Acceso a otros servicios de AWS

En este módulo avanzado, se muestra cómo los núcleos de AWS IoT Greengrass pueden interactuar con otros servicios de AWS en la nube. Este módulo se basa en el ejemplo del semáforo del Módulo 5, pero añade una función de Lambda que procesa estados de sombra y carga un resumen en una tabla de Amazon DynamoDB.



Antes de comenzar, ejecute el script de configuración de dispositivos de Greengrass o asegúrese de haber completado el módulo 1 y el módulo 2. También debe completar el módulo 5. No necesita otros componentes ni dispositivos.

Completar este módulo debería tomarle aproximadamente 30 minutos.

> **Note**
>
> En este módulo se crea y se actualiza una tabla en DynamoDB. Aunque la mayoría de las operaciones son pequeñas y entran dentro del nivel gratuito de Amazon Web Services, la ejecución de algunos de los pasos que se detallan en este módulo podría dar lugar a

cargos en su cuenta. Para obtener más información acerca de los precios, consulte la documentación de precios de DynamoDB.

Temas

- Configuración del rol del grupo
- Creación y configuración de la función de Lambda
- Configurar suscripciones
- Probar las comunicaciones

Configuración del rol del grupo

El rol del grupo es un rol de IAM que usted crea y asocia a su grupo de Greengrass. Este rol contiene los permisos que las funciones de Lambda implementadas (y otras características de AWS IoT Greengrass) utilizan para obtener acceso a los servicios de AWS. Para obtener más información, consulte the section called “Rol de grupo de Greengrass”.

Siga los siguientes pasos de alto nivel para crear un rol de grupo en la consola de IAM.

1. Cree una política que permita o deniegue acciones en uno o más recursos.
2. Cree un rol que utilice el servicio de Greengrass como una entidad de confianza.
3. Asocie su política al rol.

A continuación, en la consola AWS IoT, agregará el rol al grupo de Greengrass.

> **Note**
>
> Un grupo de Greengrass tiene un rol de grupo. Si desea añadir permisos, puede editar políticas adjuntas o adjuntar más políticas.

En este tutorial, creará una política de permisos que permita acciones de descripción, creación y actualización en una tabla de Amazon DynamoDB. A continuación, debe asociar la política a un nuevo rol y, a su vez, asociar este a su grupo de Greengrass.

En primer lugar, cree una política administrada por el cliente que conceda los permisos requeridos por la función de Lambda en este módulo.

1. En el panel de navegación de la consola de IAM, elija Políticas y después Crear política.
2. En la pestaña JSON, reemplace el contenido del marcador de posición por la política siguiente. La función de Lambda de este módulo utiliza estos permisos para crear y actualizar una tabla de DynamoDB denominada `CarStats`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionsForModule6",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:CreateTable",
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/CarStats"
        }
    ]
}
```

3. Elija Next: Tags (Siguiente: Etiquetas) y, a continuación, seleccione Next: Review (Siguiente: Revisar). En este tutorial no se utilizan etiquetas.
4. En Name (Nombre), escriba **`greengrass_CarStats_Table`** y, después, elija Create policy (Crear política).

   A continuación, cree un rol que utilice la nueva política.
5. En el panel de navegación, seleccione Roles y luego seleccione Create role.
6. En Tipo de entidad de confianza, seleccione Servicio de AWS.
7. En Caso de uso, Casos de uso para otros servicios de AWS elija Greengrass, seleccione Greengrass y, a continuación, Siguiente.
8. En Políticas de permisos, seleccione la nueva política de **`greengrass_CarStats_Table`** y, a continuación, Siguiente.
9. En Nombre del rol, ingrese **`Greengrass_Group_Role`**.

10. En Descripción, escriba **`Greengrass group role for connectors and user-defined Lambda functions`**.
11. Elija Create role (Crear rol).

    Ahora, añada el rol a su grupo de Greengrass.
12. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
13. En Grupos de Greengrass, elija su grupo.
14. Elija Configuración y, a continuación, elija Asociar rol.
15. Elija Greengrass_Group_Role de su lista de roles, y luego elija Asociar rol.

Creación y configuración de la función de Lambda

En este paso, va a crear una función de Lambda que realiza el seguimiento del número de automóviles que pasan por el semáforo. Cada vez que el estado de sombra `GG_TrafficLight` cambie a `G`, la función de Lambda simulará el paso de un número aleatorio de vehículos (de 1 a 20). Cada tercer cambio de luz a `G`, la función de Lambda envía estadísticas básicas, como los valores mínimo y máximo, a una tabla de DynamoDB.

1. En el equipo, cree una carpeta denominada `car_aggregator`.
2. Desde la carpeta de ejemplos de TrafficLight en GitHub, descargue el archivo `carAggregator.py` en la carpeta `car_aggregator`. Este es el código de la función de Lambda.

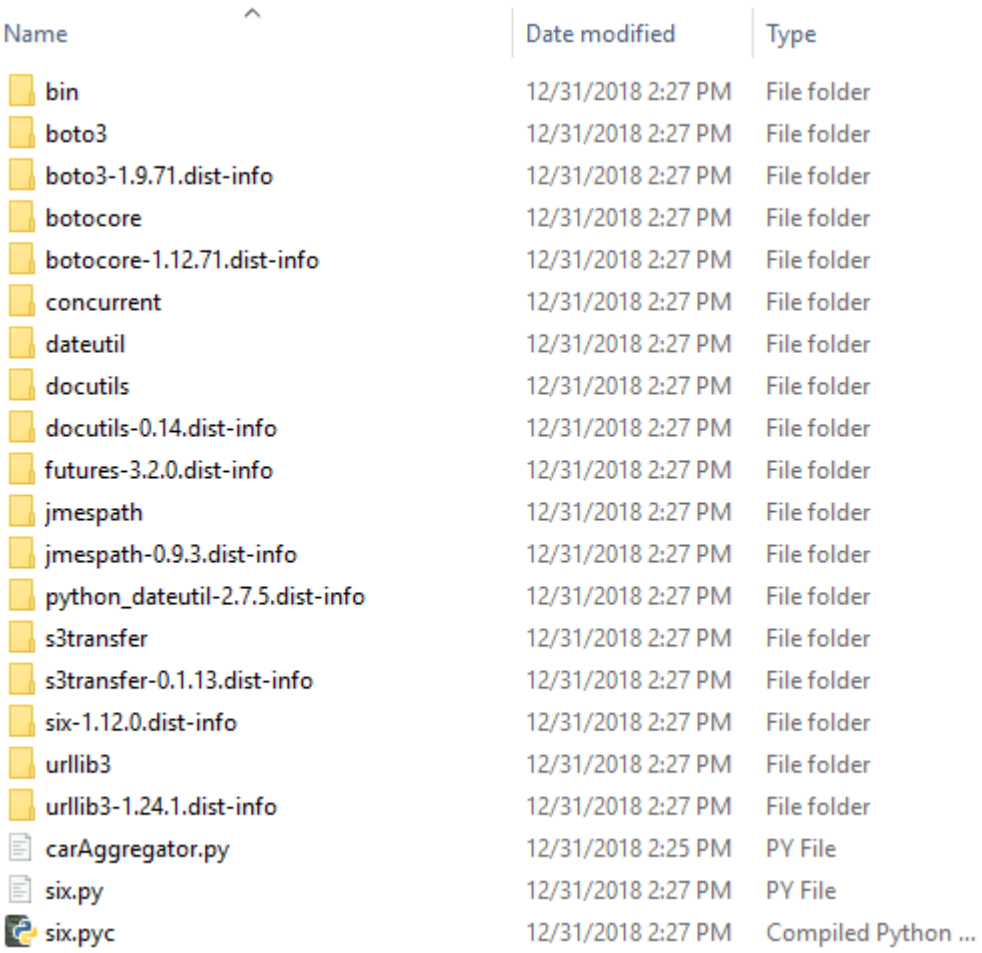
   > **Note**
   >
   > Este archivo Python de ejemplo está almacenado en el repositorio del SDK de AWS IoT Greengrass CorE por comodidad, pero no utiliza el SDK de AWS IoT Greengrass Core.
3. Si no trabaja en la región EE. UU. Este (Norte de Virgina), abra `carAggregator.py` y cambie `region_name` en la siguiente línea a la Región de AWS seleccionada actualmente en la consola AWS IoT. Para obtener las lista de las Región de AWS compatibles, consulte AWS IoT Greengrass en la Referencia general de Amazon Web Services.

   ```
   dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
   ```

4. Ejecute el siguiente comando en una ventana de línea de comandos para instalar el paquete AWS SDK for Python (Boto3) y sus dependencias en la carpeta `car_aggregator`. Las funciones de Lambda de Greengrass utilizan el SDK de AWS para obtener acceso a otros servicios de AWS. (Para Windows, utilice un símbolo del sistema elevado).

   ```
   pip install boto3 -t path-to-car_aggregator-folder
   ```

   Aparece una lista de directorios similar a la siguiente:

Name	Date modified	Type
bin	12/31/2018 2:27 PM	File folder
boto3	12/31/2018 2:27 PM	File folder
boto3-1.9.71.dist-info	12/31/2018 2:27 PM	File folder
botocore	12/31/2018 2:27 PM	File folder
botocore-1.12.71.dist-info	12/31/2018 2:27 PM	File folder
concurrent	12/31/2018 2:27 PM	File folder
dateutil	12/31/2018 2:27 PM	File folder
docutils	12/31/2018 2:27 PM	File folder
docutils-0.14.dist-info	12/31/2018 2:27 PM	File folder
futures-3.2.0.dist-info	12/31/2018 2:27 PM	File folder
jmespath	12/31/2018 2:27 PM	File folder
jmespath-0.9.3.dist-info	12/31/2018 2:27 PM	File folder
python_dateutil-2.7.5.dist-info	12/31/2018 2:27 PM	File folder
s3transfer	12/31/2018 2:27 PM	File folder
s3transfer-0.1.13.dist-info	12/31/2018 2:27 PM	File folder
six-1.12.0.dist-info	12/31/2018 2:27 PM	File folder
urllib3	12/31/2018 2:27 PM	File folder
urllib3-1.24.1.dist-info	12/31/2018 2:27 PM	File folder
carAggregator.py	12/31/2018 2:25 PM	PY File
six.py	12/31/2018 2:27 PM	PY File
six.pyc	12/31/2018 2:27 PM	Compiled Python ...

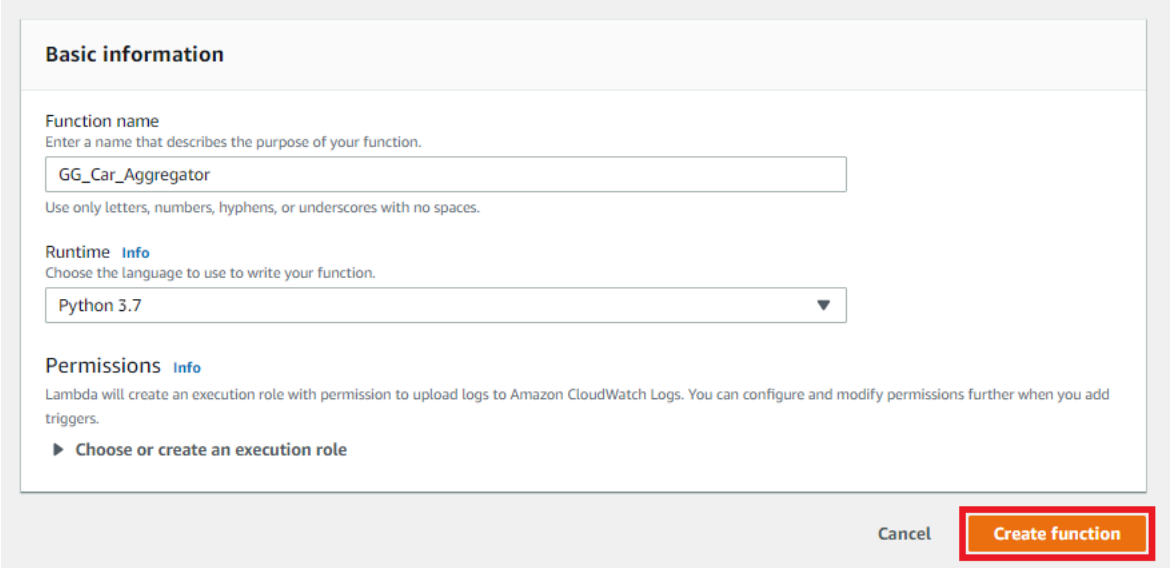
5. Comprima el contenido de la carpeta `car_aggregator` en un archivo `.zip` con el nombre `car_aggregator.zip`. (Comprima el contenido de la carpeta, no la carpeta). Este es el paquete de implementación de la función de Lambda.

6. En la consola de Lambda, cree una función denominada **`GG_Car_Aggregator`** y configure los componentes restantes del modo siguiente:

   - En Runtime (Tiempo de ejecución), elija Python 3.7.

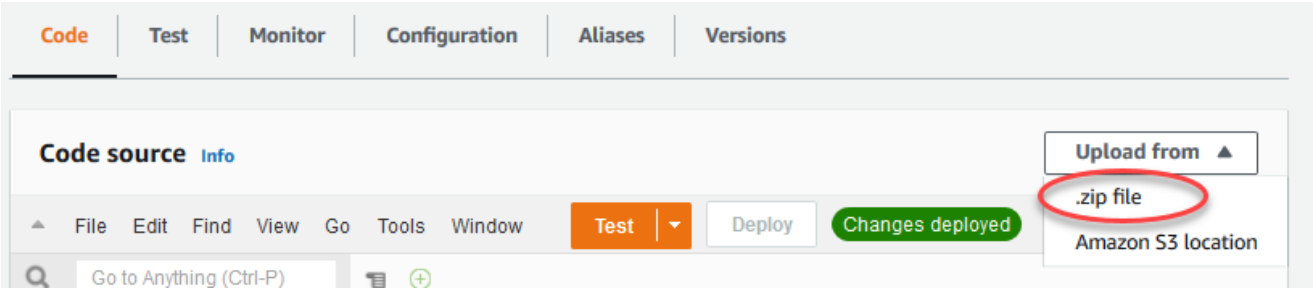
- En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol.

Elija Crear función.



7. Cargue su paquete de implementación de la función de Lambda:

   a. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip..



   b. Seleccione Cargar y, a continuación, elija su paquete de implementación `car_aggregator.zip`. A continuación, elija Guardar.

   c. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

      - En Runtime (Tiempo de ejecución), elija Python 3.7.
      - En Handler (Controlador), escriba **`carAggregator.function_handler`**.

   d. Seleccione Save.

8. Publique la función de Lambda y, a continuación, cree un alias denominado **`GG_CarAggregator`**. Para obtener instrucciones paso a paso, consulte los pasos para publicar la función de Lambda y crear un alias en el Módulo 3 (primera parte).
9. En la consola de AWS IoT, añada la función de Lambda que acaba de crear a su AWS IoT Greengrass:

   a. En la página de configuración del grupo, elija la pestaña Funciones de Lambda y, a continuación, en Mis funciones de Lambda, seleccione Añadir.

   b. Para la función de Lambda, elija GG_Car_Aggregator.

   c. Para la versión de la función de Lambda, elija el alias de la versión que publicó.

   d. En Límite de memoria, escriba **`64 MB`**.

   e. En Ancladas, elija Verdadero

   f. Elija Añadir función de Lambda.

> **Note**
>
> Puede eliminar otras funciones de Lambda de módulos anteriores.

Configurar suscripciones

En este paso, va a crear una suscripción que permita que la sombra de GG_TrafficLight envíe información actualizada sobre el estado a la función de Lambda GG_Car_Aggregator. Esta suscripción se añadirá a las suscripciones que creó en el Módulo 5, que también son necesarias en este módulo.

1. En la página de configuración de grupo, elija la pestaña Suscripciones y, a continuación, elija Agregar.
2. En la página Crear suscripción a eventos haga lo siguiente:

   a. Para Seleccionar un origen: elija Servicios y, a continuación, Servicio de sombra local.

   b. Para Tipo de destino, elija Función de lambda y, a continuación, elija GG_Car_Aggregator.

   c. En Filtro de temas, escriba **`$aws/things/GG_TrafficLight/shadow/update/documents`**.

   d. Elija Crear una suscripción.

Este módulo necesita la nueva suscripción y las suscripciones que creó en el Módulo 5.

3. Asegúrese de que el daemon de Greengrass esté en ejecución, tal y como se describe en Implementación de configuraciones de nube en un dispositivo central.
4. En la página de configuración de grupo, elija Implementar.

Probar las comunicaciones

1. En su equipo, abra dos ventanas de la línea de comando. Al igual que en el Módulo 5, una ventana se usa para el dispositivo cliente GG_Switch y la otra para el dispositivo cliente GG_TrafficLight. Utilícelas para ejecutar los mismos comandos que ejecutó en el Módulo 5.

   Ejecute los siguientes comandos para el dispositivo cliente GG_Switch:

   ```
   cd path-to-certs-folder
   python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
    --cert switchCertId-certificate.pem.crt --key switchCertId-private.pem.key --
   thingName GG_TrafficLight --clientId GG_Switch
   ```

   Ejecute los siguientes comandos para el dispositivo cliente GG_TrafficLight:

   ```
   cd path-to-certs-folder
   python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
   cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --thingName
    GG_TrafficLight --clientId GG_TrafficLight
   ```

   Cada 20 segundos, el interruptor actualiza el estado de sombra a los valores G, Y y R, y la luz indicará su nuevo estado.
2. El controlador de la función de Lambda se activa cada tres luces verdes (cada tres minutos) y se crea un nuevo registro de DynamoDB. Después de que `lightController.py` y `trafficLight.py` se hayan ejecutado durante tres minutos, vaya a la AWS Management Console y abra la consola de DynamoDB.
3. Elija Este de EE. UU. (Norte de Virginia) en el menú Región de AWS. Esta es la región en la que la función `GG_Car_Aggregator` crea la tabla.
4. En el panel de navegación, elija Tables (Tablas) y, a continuación, elija la tabla CarStats.
5. Seleccione Ver elementos para ver las entradas de la tabla.

Debería ver las entradas con estadísticas básicas con respecto a los vehículos que han pasado (una entrada cada 3 minutos). Es posible que tenga que pulsar el botón de actualización para ver las actualizaciones de la tabla.

6. Si la prueba no es correcta, puede buscar información para la solución de problemas en los registros de Greengrass.

   a. Cambie al usuario raíz y vaya al directorio `log`. El acceso a los registros de AWS IoT Greengrass requiere permisos raíz.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

   b. Compruebe si hay errores en `runtime.log`.

   ```
   cat system/runtime.log | grep 'ERROR'
   ```

   c. Compruebe el registro generado por la función de Lambda.

   ```
   cat user/region/account-id/GG_Car_Aggregator.log
   ```

   Los scripts `lightController.py` y `trafficLight.py` almacenan información de conexión en la carpeta `groupCA`, que se crea en la misma carpeta que los scripts. Si recibe errores de conexión, asegúrese de que la dirección IP del archivo `ggc-host` coincide con el punto de conexión de la dirección IP de su núcleo.

   Para obtener más información, consulte Solución de problemas.

Este es el final del tutorial básico. Ahora debería conocer el modelo de programación de AWS IoT Greengrass y sus conceptos fundamentales, incluidos los núcleos de AWS IoT Greengrass, los grupos, las suscripciones, los dispositivos y el proceso de implementación de las funciones de Lambda que se ejecutan en la periferia.

Puede eliminar la tabla de DynamoDB y las suscripciones y funciones de Lambda. Para detener las comunicaciones entre el dispositivo AWS IoT Greengrass y la nube de AWS IoT del núcelo, abra un terminal en el dispositivo del núcleo y ejecute uno de los siguientes comandos:

- Para cerrar el dispositivo de AWS IoT Greengrass del núcleo:

```
sudo halt
```

- Para detener el daemon AWS IoT Greengrass:

```
cd /greengrass/ggc/core/
sudo ./greengrassd stop
```

Módulo 7: Simulación de la integración de seguridad del hardware

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y versiones posteriores.

En este módulo avanzado se muestra cómo configurar un módulo de seguridad de hardware (HSM) simulado para su uso con un núcleo de Greengrass. La configuración utiliza SoftHSM, que es una implementación de software puro que utiliza la interfaz de programación de aplicaciones (API) PKCS#11. El objetivo de este módulo es permitirle configurar un entorno donde pueda obtener información y realizar las pruebas iniciales en una implementación de solo software de la API de PKCS#11. Se proporciona solo para el aprendizaje y las pruebas iniciales, no para su uso en producción de ningún tipo.

Puede utilizar esta configuración para experimentar usando un servicio compatible con PKCS#11 para almacenar con sus claves privadas. Para obtener más información sobre la implementación solo de software, consulte SoftHSM. Para obtener más información acerca de la integración de seguridad de hardware en un núcleo AWS IoT Greengrass, incluidos los requisitos generales, consulte the section called “Integración de la seguridad de hardware”.

> **Important**
>
> Este módulo está destinado solo a fines de experimentación. Recomendamos encarecidamente evitar el uso de SoftHSM en un entorno de producción, ya que puede proporcionar una falsa sensación de seguridad adicional. La configuración resultante no proporciona ninguna ventaja de seguridad real. Las claves almacenadas en SoftHSM no se almacenan de forma más segura que en cualquier otro medio de almacenamiento de secretos en el entorno de Greengrass.
>
> El objetivo de este módulo es permitir que aprenda acerca de la especificación de PKCS#11 y realice las pruebas iniciales de su software si pretende utilizar un HSM real basado en hardware en el futuro.

Debe probar su futura implementación de hardware por separado y por completo antes de cualquier uso en producción, ya que es posible que haya diferencias entre la implementación de PKCS#11 proporcionada en SoftHSM y una implementación basada en hardware.

Si necesita asistencia con la incorporación de un módulo de seguridad de hardware admitido, póngase en contacto con su representante de soporte de AWS Enterprise.

Antes de comenzar, ejecute el script de configuración de dispositivos de Greengrass o asegúrese de haber completado el módulo 1 y el módulo 2 del tutorial de introducción. En este módulo, suponemos que su núcleo ya está aprovisionado y comunicándose con AWS. Completar este módulo debería tomarle aproximadamente 30 minutos.

Instalación del software SoftHSM

En este paso, se instala SoftHSM y las herramientas pkcs11, que se utilizan para administrar su instancia SoftHSM.

- En un terminal en su dispositivo core AWS IoT Greengrass, ejecute el siguiente comando:

  ```
  sudo apt-get install softhsm2 libsofthsm2-dev pkcs11-dump
  ```

  Para obtener más información acerca de estos paquetes, consulte Install softhsm2, Install libsofthsm2-dev e Install pkcs11-dump.

  > **Note**
  >
  > Si tiene problemas al utilizar este comando en su sistema, consulte SoftHSM version 2 en GitHub. Este sitio proporciona más información de la instalación, incluido cómo compilar desde el origen.

Configurar SoftHSM

En este paso, configurará SoftHSM.

1. Cambie al usuario raíz.

```
sudo su
```

2. Utilice la página del manual para encontrar la ubicación `softhsm2.conf` en el sistema. Una ubicación común es `/etc/softhsm/softhsm2.conf`, pero la ubicación puede ser diferente en algunos sistemas.

```
man softhsm2.conf
```

3. Cree el directorio para el archivo de configuración softhsm2 en la ubicación del sistema. En este ejemplo, supondremos que la ubicación es `/etc/softhsm/softhsm2.conf`.

```
mkdir -p /etc/softhsm
```

4. Cree el directorio del token en el directorio `/greengrass`.

> **Note**
>
> Si se omite este paso, softhsm2-util muestra `ERROR: Could not initialize the library`.

```
mkdir -p /greengrass/softhsm2/tokens
```

5. Configure el directorio del token.

```
echo "directories.tokendir = /greengrass/softhsm2/tokens" > /etc/softhsm/
softhsm2.conf
```

6. Configure un backend basado en archivos.

```
echo "objectstore.backend = file" >> /etc/softhsm/softhsm2.conf
```

> **Note**
>
> Estos valores de configuración son solo con fines de experimentación. Para ver todas las opciones de configuración, lea la página del manual del archivo de configuración.

```
man softhsm2.conf
```

Importación de la clave privada en SoftHSM.

En este paso, debe inicializar el token de SoftHSM, convertir el formato de clave privada y, a continuación, importar la clave privada.

1. Inicializar el token de SoftHSM.

   ```
   softhsm2-util --init-token --slot 0 --label greengrass --so-pin 12345 --pin 1234
   ```

   > **Note**
   >
   > Si se le pide, introduzca el pin de SO `12345` y el pin de usuario `1234`. AWS IoT Greengrass no utiliza el pin de SO (supervisor), por lo que puede utilizar cualquier valor. Si recibe el error `CKR_SLOT_ID_INVALID: Slot 0 does not exist`, pruebe el siguiente comando en su lugar:
   >
   > ```
   > softhsm2-util --init-token --free --label greengrass --so-pin 12345 --pin
   >  1234
   > ```

2. Convierta la clave privada en un formato que se pueda utilizar por la herramienta de importación de SoftHSM. Para este tutorial, puede convertir la clave privada que se obtuvo de la opción Default Group creation (Creación predeterminada de grupo) en el Módulo 2 del tutorial de introducción.

   ```
   openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in hash.private.key -
   out hash.private.pem
   ```

3. Importar la clave privada a SoftHSM. Ejecute solo uno de los siguientes comandos, en función de su versión de softhsm2-util.

   Sintaxis de Raspbian softhsm2-util v2.2.0

   ```
   softhsm2-util --import hash.private.pem --token greengrass --label iotkey --id
    0000 --pin 12340
   ```

Sintaxis de Ubuntu softhsm2-util v2.0.0

```
softhsm2-util --import hash.private.pem --slot 0 --label iotkey --id 0000 --pin
 1234
```

Este comando identifica el slot como `0` y define la etiqueta de clave como `iotkey`. Usará estos valores en la sección siguiente.

Después de que se importe la clave privada, tiene la opción de quitarla del directorio `/greengrass/certs`. Asegúrese de mantener los certificados CA raíz y de dispositivo en el directorio.

Configuración del núcleo de Greengrass para que utilice SoftHSM

En este paso, debe modificar el archivo de configuración del núcleo de Greengrass para utilizar SoftHSM.

1. Busque la ruta de la biblioteca del proveedor de SoftHSM (`libsofthsm2.so`) en el sistema:

   a. Obtenga la lista de los paquetes instalados para la biblioteca.

   ```
   sudo dpkg -L libsofthsm2
   ```

   El archivo `libsofthsm2.so` se encuentra en el directorio `softhsm`.

   b. Copie la ruta completa al archivo (por ejemplo, `/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so`). Usará este valor más tarde.

2. Detenga el daemon de Greengrass.

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd stop
   ```

3. Abra el archivo de configuración de Greengrass. Este es el archivo `config.json` del directorio `/greengrass/config`.

> **Note**
>
> Los ejemplos en este procedimiento se han escrito partiendo del supuesto de que el archivo `config.json` utiliza el formato generado desde la opción Default Group Creation (Creación predeterminada de grupo) en el Módulo 2 del tutorial de introducción.

4. En el objeto `crypto.principals`, inserte el siguiente objeto de certificado de servidor MQTT. Añada una coma cuando sea necesario para crear un archivo JSON válido.

```
"MQTTServerCertificate": {
  "privateKeyPath": "path-to-private-key"
}
```

5. En el objeto `crypto`, inserte el siguiente objeto PKCS11. Añada una coma cuando sea necesario para crear un archivo JSON válido.

```
"PKCS11": {
  "P11Provider": "/path-to-pkcs11-provider-so",
  "slotLabel": "crypto-token-name",
  "slotUserPin": "crypto-token-user-pin"
}
```

El archivo debe ser similar al siguiente:

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix.iot.region.amazonaws.com",
    "ggHost" : "greengrass.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
```

```
  "crypto": {
    "PKCS11": {
      "P11Provider": "/path-to-pkcs11-provider-so",
      "slotLabel": "crypto-token-name",
      "slotUserPin": "crypto-token-user-pin"
    },
    "principals" : {
      "MQTTServerCertificate": {
        "privateKeyPath": "path-to-private-key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      },
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

> **Note**
>
> Para utilizar las actualizaciones inalámbrica (OTA) con la seguridad por hardware, el objeto `PKCS11` debe contener también la propiedad `OpenSSLEngine`. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.

6. Edite el objeto `crypto`:

   a. Configure el objeto `PKCS11`.

      - En `P11Provider`, introduzca la ruta completa a `libsofthsm2.so`.
      - En `slotLabel`, introduzca `greengrass`.
      - En `slotUserPin`, introduzca `1234`.

   b. Configure las rutas de clave privada en el objeto `principals`. No edite la propiedad `certificatePath`.

- Para las propiedades de `privateKeyPath`, escriba la siguiente ruta RFC 7512 PKCS#11 (que especifica la etiqueta de la clave). Haga esto para los principales de `IoTCertificate`, `SecretsManager` y `MQTTServerCertificate`.

```
pkcs11:object=iotkey;type=private
```

c. Compruebe el objeto `crypto`. Debería parecerse a lo que sigue:

```
"crypto": {
  "PKCS11": {
    "P11Provider": "/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so",
    "slotLabel": "greengrass",
    "slotUserPin": "1234"
  },
  "principals": {
    "MQTTServerCertificate": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "SecretsManager": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "IoTCertificate": {
      "certificatePath": "file://certs/core.crt",
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    }
  },
  "caPath": "file://certs/root.ca.pem"
}
```

7. Elimine los valores `caPath`, `certPath` y `keyPath` del objeto `coreThing`. Debería parecerse a lo que sigue:

```
"coreThing" : {
  "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
  "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
  "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
  "keepAlive" : 600
}
```

> **Note**
>
> Para este tutorial, debe especificar la misma clave privada para todos los principales. Para obtener más información acerca de cómo elegir la clave privada del servidor MQTT local, consulte Performance. Para obtener más información sobre el Secrets Manager local, consulte Implementación de secretos en el núcleo.

Probar la configuración

- Iniciar el daemon de Greengrass.

  ```
  cd /greengrass/ggc/core/
  sudo ./greengrassd start
  ```

  Si el daemon se inicia correctamente, su núcleo se ha configurado correctamente.

  Ahora está listo para obtener más información acerca de la especificación de PKCS#11 y realizar las pruebas iniciales con la API de PKCS#11 que se proporciona en la implementación de SoftHSM.

  > **Important**
  >
  > De nuevo, es muy importante tener en cuenta que este módulo está diseñado para el aprendizaje y las pruebas únicamente. En realidad no aumenta la seguridad general del entorno de Greengrass.
  > En lugar de ello, el objetivo del módulo es permitirle comenzar a aprender y realizar pruebas como preparación para utilizar un auténtico HSM basado en hardware en el futuro. Todo ese tiempo, debe probar su software frente a HSM basado en hardware por separado y por completo antes de cualquier uso en producción, ya que es posible que haya diferencias entre la implementación de PKCS#11 proporcionada en SoftHSM y una implementación basada en hardware.

Véase también

- PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40. Editado por John Leiseboer y Robert Griffin. 16 de noviembre de 2014. OASIS Committee Note 02. http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html. Última versión: http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.
- RFC 7512

Actualizaciones de OTA para el software AWS IoT Greengrass Core

El paquete del software AWS IoT Greengrass Core incluye un agente de actualización que puede realizar actualizaciones transparentes (OTA) del software AWS IoT Greengrass. Puede utilizar las actualizaciones OTA para instalar la última versión del software AWS IoT Greengrass Core o del software de agente de actualización OTA en uno o más núcleos. Con las actualizaciones OTA, no es necesario que los dispositivos del núcleo estén físicamente presentes.

Le recomendamos que utilice las actualizaciones OTA cuando sea posible. Proporcionan un mecanismo que puede utilizar para realizar un seguimiento del estado de la actualización y del historial de actualizaciones. Si se produce un error en la actualización, el agente de actualización de OTA vuelve a la versión de software anterior.

> **Note**
>
> Las actualizaciones OTA no son compatibles cuando se utiliza `apt` para instalar el software AWS IoT Greengrass Core. En estas instalaciones, le recomendamos que utilice `apt` para actualizar el software. Para obtener más información, consulte the section called “Instalación desde un repositorio de APT”.

Las actualizaciones OTA hacen que sea más eficiente:

- Corrige vulnerabilidades de seguridad.
- Solucionar problemas de estabilidad del software.
- Implementar características nuevas o mejoradas.

Esta característica se integra con trabajos de AWS IoT.

Requisitos

Los siguientes requisitos se aplican a las actualizaciones OTA del software AWS IoT Greengrass.

- El núcleo de Greengrass debe tener al menos 400 MB de espacio en disco disponible en el almacenamiento local. El agente de actualización de OTA requiere aproximadamente tres veces el

requisito de uso de tiempo de ejecución del software AWS IoT Greengrass Core. Para obtener más información, consulte Cuotas de servicio para el núcleo de Greengrass en la Referencia general de Amazon Web Services.

- El núcleo de Greengrass debe tener una conexión con la Nube de AWS.
- El núcleo de Greengrass debe configurarse y aprovisionarse correctamente con certificados y claves para la autenticación con AWS IoT Core y AWS IoT Greengrass. Para obtener más información, consulte the section called “Certificados X.509”.
- El núcleo de Greengrass no se puede configurar para usar un proxy de red.

> **Note**
>
> A partir de la versión 1.9.3 de AWS IoT Greengrass, las actualizaciones OTA son compatibles con los núcleos que configuran el tráfico MQTT para utilizar el puerto 443 en lugar del puerto predeterminado 8883. Sin embargo, el agente de actualización de OTA no admite actualizaciones a través de un proxy de red. Para obtener más información, consulte the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.

- No se puede habilitar el arranque de confianza en la partición que contiene el software AWS IoT Greengrass Core.

> **Note**
>
> Puede instalar y ejecutar el software AWS IoT Greengrass Core en una partición con arranque de confianza habilitado, pero las actualizaciones OTA no se admiten.

- AWS IoT Greengrass debe tener permisos de lectura/escritura en la partición que contiene el software AWS IoT Greengrass Core.
- Si utiliza un sistema init para administrar su núcleo de Greengrass, debe configurar las actualizaciones OTA para que se integren con el sistema init. Para obtener más información, consulte the section called “Integración con sistemas init”.
- Debe crear un rol que se utilice para prefirmar las URL de Amazon S3 para los artefactos de actualización del software AWS IoT Greengrass. Este rol de firmante permite a AWS IoT Core acceder a artefactos de actualización de software almacenados en Amazon S3 en su nombre. Para obtener más información, consulte the section called “Permisos de IAM para actualizaciones OTA”.

Permisos de IAM para actualizaciones OTA

Cuando AWS IoT Greengrass lanza una nueva versión del software AWS IoT Greengrass Core, AWS IoT Greengrass actualiza los artefactos de software almacenados en Amazon S3 que se utilizan para la actualización OTA.

Su cuenta de Cuenta de AWS debe incluir un rol de firmante de URL de Amazon S3 que pueda utilizarse para acceder a estos artefactos. El rol debe tener una política de permisos que permita la acción `s3:GetObject` en los buckets de las Región de AWS de destino. El rol también debe tener una política de confianza que permita a `iot.amazonaws.com` asumir el rol como una entidad de confianza.

Política de permisos

Para los permisos de rol, puede utilizar la política administrada de AWS o crear una política personalizada.

- Utilizar la política administrada de AWS

  La política administrada GreenGrassotaupdateartActAccess la proporciona AWS IoT Greengrass. Utilice esta política si desea permitir el acceso en todas las regiones de Amazon Web Services admitidas por AWS IoT Greengrass, tanto actuales como futuras.

- Crear una política personalizada

  Debe crear una política personalizada si desea especificar explícitamente en qué regiones de Amazon Web Services se implementan sus núcleos. La siguiente política de ejemplo permite el acceso a actualizaciones de software de AWS IoT Greengrass en seis regiones.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToGreengrassOTAUpdateArtifacts",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::us-east-1-greengrass-updates/*",
                "arn:aws:s3:::us-west-2-greengrass-updates/*",
                "arn:aws:s3:::ap-northeast-1-greengrass-updates/*",
```

```
                "arn:aws:s3:::ap-southeast-2-greengrass-updates/*",
                "arn:aws:s3:::eu-central-1-greengrass-updates/*",
                "arn:aws:s3:::eu-west-1-greengrass-updates/*"
            ]
        }
    ]
}
```

Política de confianza

La política de confianza asociada al rol debe permitir la acción `sts:AssumeRole` y definir `iot.amazonaws.com` como principal. Esto permite a AWS IoT Core asumir el rol como una entidad de confianza. A continuación se muestra un ejemplo de documento de política.

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

Además, el usuario que inicia una actualización OTA debe tener permisos para usar `greengrass:CreateSoftwareUpdateJob` y `iot:CreateJob`, y para usar `iam:PassRole` para transferir los permisos del rol de firmante. A continuación se muestra un ejemplo de política de IAM.

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

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJob"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn-of-s3-url-signer-role"
        }
    ]
}
```

Consideraciones

Antes de lanzar una actualización OTA del software Greengrass Core, tenga en cuenta el impacto en los dispositivos de su grupo de Greengrass, tanto en el dispositivo del núcleo como en los dispositivos cliente conectados localmente a ese dispositivo del núcleo:

- El dispositivo principal se cierra durante la actualización.
- Las funciones Lambda que se ejecuten en el dispositivo principal se cerrarán. Si estas funciones escriben en recursos locales, es posible que dejen esos recursos en un estado incorrecto, salvo que se cierren correctamente.
- Durante el tiempo de inactividad del dispositivo principal, se perderán todas sus conexiones con la Nube de AWS. Los mensajes enviados a través del dispositivo principal por los dispositivos cliente se perderán.
- Las cachés de credenciales se perderán.
- Las colas que mantienen los trabajos pendientes de las funciones Lambda se perderán.
- Las funciones de Lambda de larga duración perderán la información de estado dinámico y se eliminarán todos los trabajos pendientes.

La siguiente información de estado se guarda durante una actualización OTA:

- Configuración de Core

- Configuración del grupo de Greengrass
- Sombras locales
- Registros de Greengrass
- Registros del agente de actualización de OTA

Agente de actualización de OTA para Greengrass

El agente de actualización de OTA de Greengrass es el componente de software en el dispositivo, que se encarga de actualizar los trabajos creados e implementados en la nube. El agente de actualización de OTA se distribuye en el mismo paquete de software que el software AWS IoT Greengrass Core. El agente se encuentra en */greengrass-root*`/ota/ota_agent/ggc-ota`. Escribe registros en `/var/log/greengrass/ota/ggc_ota.txt`.

> **Note**
>
> */greengrass-root* representa la ruta donde está instalado el software de AWS IoT Greengrass Core en su dispositivo. Normalmente, este es el directorio `/greengrass`.

Puede iniciar el agente de actualización de OTA mediante la ejecución manual del binario o integrándolo como parte de un script init, por ejemplo, un archivo del servicio systemd. Si ejecuta el binario manualmente, debe ejecutarse como raíz. Cuando se inicia, el agente de actualización de OTA escucha los trabajos de actualización del software AWS IoT Greengrass desde AWS IoT Core y los ejecuta secuencialmente. El agente de actualización de OTA ignora todos los demás tipos de trabajo de AWS IoT.

El siguiente extracto muestra un ejemplo de un archivo de servicio systemd para iniciar, detener y reiniciar el agente de actualización de OTA:

```
[Unit]
Description=Greengrass OTA Daemon

[Service]
Type=forking
Restart=on-failure
ExecStart=/greengrass/ota/ota_agent/ggc-ota
```

```
[Install]
WantedBy=multi-user.target
```

Un núcleo que sea el destino de una actualización no debe ejecutar dos instancias del agente de actualización de OTA. Si lo hace, los dos agentes procesarán los mismos trabajos, lo que causará conflictos.

Integración con sistemas init

Durante una actualización OTA, el agente de actualización de OTA reinicia los binarios en el núcleo. Si los binarios se están ejecutando, esto puede provocar conflictos cuando un sistema init está supervisando el estado del software AWS IoT Greengrass Core o del agente durante la actualización. Para ayudarle a integrar el mecanismo de actualización OTA con sus estrategias de monitorización init, puede escribir scripts de shell que se ejecuten antes y después de una actualización. Por ejemplo, puede utilizar el script `ggc_pre_update.sh` para realizar copias de seguridad de los datos o detener procesos antes de que el dispositivo se apague.

Para indicar al agente de actualización de OTA que ejecute estos scripts, debe incluir la marca `"managedRespawn" : true` en el archivo config.json. Este ajuste se muestra en el siguiente fragmento:

```
{
   "coreThing": {
       …
   },
   "runtime": {
       …
   },
   "managedRespawn": true
       …
}
```

Regeneración administrada con actualizaciones OTA

Los siguientes requisitos se aplican a las actualizaciones de OTA con `managedRespawn` establecidas en `true`:

- Los siguientes scripts de shell deben estar presentes en el directorio */greengrass-root*`/usr/scripts`:

  - `ggc_pre_update.sh`
  - `ggc_post_update.sh`
  - `ota_pre_update.sh`
  - `ota_post_update.sh`
- Los scripts deben devolver un código de devolución correcto.
- Los scripts deben ser propiedad de la raíz y deben ser ejecutables desde la raíz únicamente.
- El script `ggc_pre_update.sh` debe detener al daemon de Greengrass.
- El script `ggc_post_update.sh` debe detener al daemon de Greengrass.

> **Note**
>
> Como el agente de actualización OTA gestiona su propio proceso, no es necesario que los scripts `ota_pre_update.sh` y `ota_post_update.sh` detengan ni inicien el servicio OTA.

El agente de actualización OTA ejecuta los scripts desde */greengrass-root*`/usr/scripts`. El árbol de directorios debe ser similar al siguiente:

```
<greengrass_root>
|-- certs
|-- config
|   |-- config.json
|-- ggc
|-- usr/scripts
|   |-- ggc_pre_update.sh
|   |-- ggc_post_update.sh
|   |-- ota_pre_update.sh
|   |-- ota_post_update.sh
|-- ota
```

Si `managedRespawn` se establece en `true`, el agente de actualización de OTA comprueba el directorio */greengrass-root*`/usr/scripts` de los scripts antes y después de la actualización del software. Si los scripts no existen, se produce un error en la actualización. AWS IoT Greengrass no valida el contenido de estos scripts. Como práctica recomendada, compruebe que los scripts funcionan correctamente y emita los códigos de salida adecuados para detectar errores.

Para las actualizaciones OTA del software AWS IoT Greengrass Core:

- Antes de iniciar la actualización, el agente ejecuta el script `ggc_pre_update.sh`. Utilice este script para los comandos que deben ejecutarse antes de que el agente de actualización OTA inicie la actualización del software de AWS IoT Greengrass Core, por ejemplo, para hacer copias de seguridad de los datos o detener cualquier proceso en ejecución. En el ejemplo siguiente se muestra una secuencia de comandos simple para detener el daemon Greengrass.

  ```
  #!/bin/bash
  set -euo pipefail
  systemctl stop greengrass
  ```

- Después de completar la actualización, el agente ejecuta el script `ggc_post_update.sh`. Utilice este script para comandos que deban ejecutarse después de que el agente de actualización OTA inicie la actualización del software de AWS IoT Greengrass Core, como para reiniciar procesos. En el ejemplo siguiente se muestra una secuencia de comandos simple para iniciar el daemon Greengrass.

  ```
  #!/bin/bash
  set -euo pipefail
  systemctl start greengrass
  ```

Para las actualizaciones OTA del agente de actualización OTA:

- Antes de iniciar la actualización, el agente ejecuta el script `ota_pre_update.sh`. Utilice este script para comandos que deban ejecutarse antes de que el agente de actualización OTA se actualice a sí mismo, como para realizar copias de seguridad de los datos o detener cualquier proceso en ejecución.
- Después de completar la actualización, el agente ejecuta el script `ota_post_update.sh`. Utilice este script para comandos que deban ejecutarse después de que el agente de actualización OTA se actualice a sí mismo, como para reiniciar procesos.

> **Note**
>
> Si `managedRespawn` se establece en `false`, el agente de actualización de OTA no ejecuta los scripts.

Crear una actualización OTA

Siga estos pasos para realizar una actualización OTA del software AWS IoT Greengrass en uno o más núcleos:

1. Asegúrese de que los núcleos cumplan los requisitos de las actualizaciones OTA.

   > **Note**
   >
   > Si configuró un sistema init para administrar el software de AWS IoT Greengrass Core o el agente de actualización de OTA, compruebe lo siguiente en los núcleos:
   >
   > - El archivo config.json especifica `"managedRespawn" : true`.
   > - El directorio /*greengrass-root*/usr/scripts contiene los siguientes scripts:
   >   - ggc_pre_update.sh
   >   - ggc_post_update.sh
   >   - ota_pre_update.sh
   >   - ota_post_update.sh
   >
   > Para obtener más información, consulte the section called “Integración con sistemas init”.

2. En un terminal de dispositivo del núcleo, inicie el agente de actualización de OTA.

   ```
   cd /greengrass-root/ota/ota_agent
   sudo ./ggc-ota
   ```

   > **Note**
   >
   > *greengrass-root* representa la ruta donde está instalado el software de AWS IoT Greengrass Core en su dispositivo. Normalmente, este es el directorio `/greengrass`.

   No inicie varias instancias del agente de actualización de OTA en un núcleo porque podría causar conflictos.

3. Utilice la API de AWS IoT Greengrass para crear un trabajo de actualización de software.

a. Llamada a la API CreateSoftwareUpdateJob En este procedimiento de ejemplo, usamos comandos de la AWS CLI.

   El siguiente comando crea un trabajo que actualiza el software AWS IoT Greengrass Core en un núcleo. Reemplace los valores de ejemplo y, a continuación, ejecute el comando.

   Linux or macOS terminal

   ```
   aws greengrass create-software-update-job \
   --update-targets-architecture x86_64 \
   --update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] \
   --update-targets-operating-system ubuntu \
   --software-to-update core \
   --s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
   --update-agent-log-level WARN \
   --amzn-client-token myClientToken1
   ```

   Windows command prompt

   ```
   aws greengrass create-software-update-job ^
   --update-targets-architecture x86_64 ^
   --update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] ^
   --update-targets-operating-system ubuntu ^
   --software-to-update core ^
   --s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole ^
   --update-agent-log-level WARN ^
   --amzn-client-token myClientToken1
   ```

   El comando devuelve el siguiente resultado.

   ```
   {
       "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
       "IotJobArn": "arn:aws:iot:region:123456789012:job/
   GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
       "PlatformSoftwareVersion": "1.10.1"
   }
   ```

b. Copie el `IoTJobId` de la respuesta.

c. Llame a DescribeJob en la API de AWS IoT Core para ver el estado del trabajo. Reemplace el valor de ejemplo con su ID de trabajo y, a continuación, ejecute el comando.

```
aws iot describe-job --job-id GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-
a1da0EXAMPLE
```

El comando devuelve un objeto de respuesta que contiene información sobre el trabajo, incluidos `status` y `jobProcessDetails`.

```
{
    "job": {
        "jobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
        "jobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
        "targetSelection": "SNAPSHOT",
        "status": "IN_PROGRESS",
        "targets": [
            "arn:aws:iot:region:123456789012:thing/myCoreDevice"
        ],
        "description": "This job was created by Greengrass to update the
 Greengrass Cores in the targets with version 1.10.1 of the core software
 running on x86_64 architecture.",
        "presignedUrlConfig": {
            "roleArn": "arn:aws::iam::123456789012:role/myS3UrlSignerRole",
            "expiresInSec": 3600
        },
        "jobExecutionsRolloutConfig": {},
        "createdAt": 1588718249.079,
        "lastUpdatedAt": 1588718253.419,
        "jobProcessDetails": {
            "numberOfCanceledThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfFailedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfQueuedThings": 1,
            "numberOfInProgressThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfTimedOutThings": 0
        },
        "timeoutConfig": {}
    }
```

```
}
```

Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

API CreateSoftwareUpdateJob

Puede utilizar la API de `CreateSoftwareUpdateJob` para actualizar el software de AWS IoT Greengrass Core o el agente de actualización de OTA en los dispositivos del núcleo. Esta API crea un trabajo de instantánea de AWS IoT que notifica a los dispositivos cuando hay una actualización disponible. Después de llamar a `CreateSoftwareUpdateJob`, puede utilizar otros comandos de trabajo de AWS IoT para realizar un seguimiento de la actualización de software. Para obtener más información, consulte Empleos en la Guía para desarrolladores de AWS IoT.

En el siguiente ejemplo se muestra cómo usar la AWS CLI para crear un trabajo que actualiza el software AWS IoT Greengrass Core en un dispositivo del núcleo:

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
--update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] \
--update-targets-operating-system ubuntu \
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

El comando `create-software-update-job` devuelve una respuesta JSON que contiene el ID de trabajo, el ARN del trabajo y la versión de software instalada por la actualización:

```
{
    "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "IotJobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-
ee80-4d42-8321-a1da0EXAMPLE",
    "PlatformSoftwareVersion": "1.9.2"
}
```

Para ver los pasos que muestran cómo utilizar `create-software-update-job` para actualizar un dispositivo del núcleo, consulte the section called “Crear una actualización OTA”.

El comando `create-software-update-job` tiene los parámetros siguientes:

`--update-targets-architecture`

La arquitectura del dispositivo del núcleo.

Valores válidos: `armv7l`, `armv6l`, `x86_64` o `aarch64`

`--update-targets`

Los núcleos que se van a actualizar. La lista puede contener ARN de núcleos individuales y ARN de grupos de objetos cuyos miembros son núcleos. Para obtener más información acerca de los grupos de objetos, consulte Grupos de objetos estáticos en la Guía para desarrolladores de AWS IoT.

`--update-targets-operating-system`

El sistema operativo del dispositivo del núcleo.

Valores válidos: `ubuntu`, `amazon_linux`, `raspbian` o `openwrt`

`--software-to-update`

Especifica si el software del dispositivo principal o el software del agente de actualización de OTA deberían actualizarse.

Valores válidos: `core` o `ota_agent`

`--s3-url-signer-role`

El ARN del rol de IAM que se utiliza para prefirmar la URL de Amazon S3 que enlaza con los artefactos de actualización del software AWS IoT Greengrass. La política de permisos asociada al rol debe permitir la acción `s3:GetObject` en los buckets de las Región de AWS de destino. El rol también debe permitir a `iot.amazonaws.com` asumir el rol como una entidad de confianza. Para obtener más información, consulte the section called “Permisos de IAM para actualizaciones OTA”.

`--amzn-client-token`

(Opcional) Un token de cliente utilizado para realizar solicitudes idempotentes. Proporcione un token exclusivo para evitar que se creen actualizaciones duplicadas debido a reintentos internos.

`--update-agent-log-level`

(Opcional) El nivel de registro para las instrucciones de registro generadas por el agente de actualización de OTA. El valor predeterminado es `ERROR`.

Valores válidos: `NONE`, `TRACE`, `DEBUG`, `VERBOSE`, `INFO`, `WARN`, `ERROR` o `FATAL`

Note

`CreateSoftwareUpdateJob` acepta solicitudes solo para las siguientes combinaciones de arquitectura y sistema operativo compatibles:

- ubuntu/x86_64
- ubuntu/aarch64
- amazon_linux/x86_64
- raspbian/armv7l
- raspbian/armv6l
- openwrt/aarch64
- openwrt/armv7l

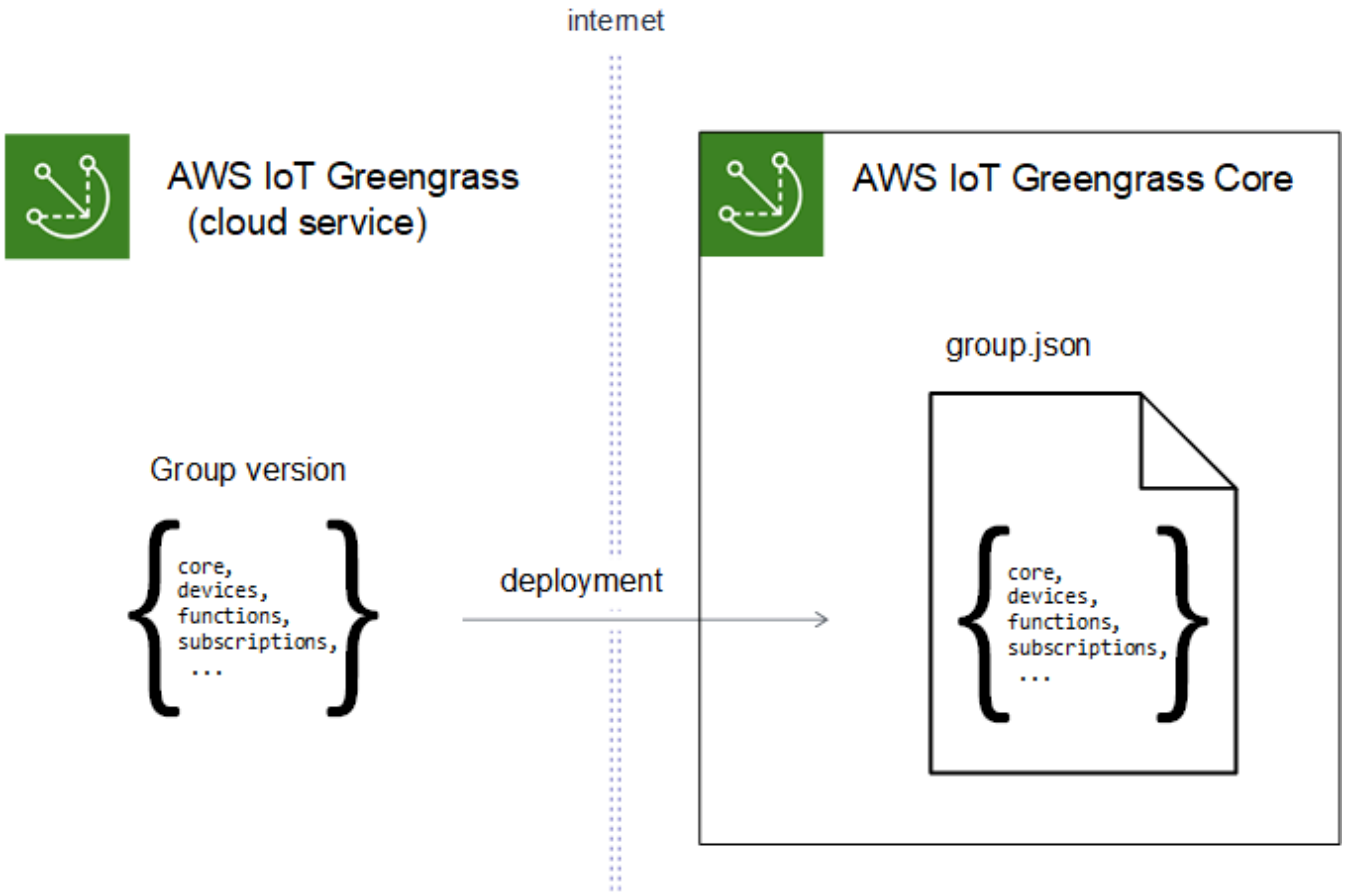
Implementación de grupos de AWS IoT Greengrass en un núcleo de AWS IoT Greengrass

Use los grupos de AWS IoT Greengrass para organizar entidades en el entorno perimetral. También puede utilizar los grupos para controlar cómo interactúan las entidades del grupo entre sí y con el Nube de AWS. Por ejemplo, solo las funciones de Lambda del grupo se implementan para ejecutarse de manera local y solo los dispositivos del grupo pueden comunicarse mediante el servidor MQTT local.

Un grupo debe incluir un núcleo, que es un dispositivo de AWS IoT que ejecuta el software AWS IoT Greengrass Core. El núcleo actúa como una gateway perimetral y proporciona funcionalidades de AWS IoT Core en el entorno perimetral. En función de las necesidades de su negocio, también puede añadir a un grupo las siguientes entidades:

- Dispositivos cliente. Se representan como objetos en el registro de AWS IoT. Estos dispositivos deben ejecutar FreeRTOS o deben utilizar el SDK de dispositivos de AWS IoT o la API de detección de AWS IoT Greengrass para obtener la información de conexión del núcleo. Solo los dispositivos cliente que son miembros del grupo pueden conectarse al núcleo.
- Funciones de Lambda Son aplicaciones definidas por los usuarios sin servidor que ejecutan código en el núcleo. Las funciones de Lambda se crean en AWS Lambda y se hace referencia a ellas desde un grupo de Greengrass. Para obtener más información, consulte Ejecutar funciones de Lambda locales .
- Conectores. Son aplicaciones predefinidas sin servidor que ejecutan código en el núcleo. Los conectores pueden integrarse con la infraestructura local, con protocolos de dispositivos, con AWS y con otros servicios en la nube. Para obtener más información, consulte Integración con servicios y protocolos mediante conectores.
- Suscripciones. Definen los publicadores, los suscriptores y los temas (o asuntos) de MQTT que están autorizados para la comunicación con MQTT.
- Recursos. Referencias a dispositivos y volúmenes locales, modelos de machine learning y secretos, que se utilizan para controlar el acceso por parte de Greengas en funciones y conectores de Lambda.
- Registros. Son configuraciones de registro de los componentes del sistema de AWS IoT Greengrass y las funciones de Lambda. Para obtener más información, consulte the section called “Monitorización con registros de AWS IoT Greengrass”.

Puede administrar el grupo de Greengrass en la Nube de AWS e implementarlo después en un núcleo. La implementación copia la configuración del grupo en el archivo `group.json` del dispositivo del núcleo. Este archivo se encuentra en *`greengrass-root`*`/ggc/deployments/group`.



> **Note**
>
> Durante una implementación, el proceso del demonio de Greengrass del dispositivo del núcleo se detiene y después se reinicia.

Implementación de grupos desde la consola de AWS IoT

Puede implementar un grupo y administrar sus implementaciones desde la página de configuración del grupo de la consola AWS IoT.

> **Note**
>
> Para abrir esta página en la consola, seleccione Dispositivos Greengrass, luego Grupos (V1) y, a continuación, en Grupos de Greengrass, seleccione su grupo.

Para implementar la versión actual del grupo

- Desde la página de configuración de grupo, elija Implementaciones.

Para ver el historial de implementaciones del grupo

El historial de implementaciones de un grupo incluye la fecha y la hora, la versión del grupo y el estado de cada intento de implementación.

1. Desde la página de configuración de grupo, elija la pestaña Implementaciones.
2. Para ver más información sobre una implementación, incluidos los mensajes de error, seleccione Implementaciones en la consola AWS IoT, en Dispositivos de Greengrass.

Para volver a implementar una implementación de grupo

Es posible que desee repetir una implementación si se produce un error en la implementación actual o si quiere volver a otra versión del grupo.

1. Desde la consola de AWS IoT, seleccione dispositivos de Greengrass y, a continuación, Grupos (V1).
2. Elija la pestaña Implementaciones.
3. Elija la implementación que desea volver a implementar y elija Reimplementar.

Para restablecer las implementaciones de grupo

Es posible que desee restablecer las implementaciones de grupo para mover o eliminar un grupo o para eliminar la información de implementación. Para obtener más información, consulte the section called “Restablecimiento de implementaciones”.

1. Desde la consola de AWS IoT, seleccione dispositivos de Greengrass y, a continuación, Grupos (V1).

2. Elija la pestaña Implementaciones.
3. Elija la implementación que desea restablecer y elija Restablecer implementaciones.

Implementación de grupos con la API de AWS IoT Greengrass

La API de AWS IoT Greengrass proporciona las siguientes acciones para implementar grupos de AWS IoT Greengrass y administrar implementaciones de grupo. Puede llamar a estas acciones desde la AWS CLI, la API de AWS IoT Greengrass o el SDK de AWS.

Acción de	Descripción
CreateDeployment	Crea una implementación `Redeployment` o `NewDeployment` . Es posible que desee repetir una implement ación si se produce un error en la implement ación actual. O es posible que desee repetirla para revertir a otra versión del grupo.
GetDeploymentStatus	Devuelve el estado de una implementación: `Building`, `InProgress` , `Success` o `Failure`. Puedes configurar EventBridge los eventos de Amazon para recibir notificaciones de despliegue. Para obtener más información, consulte the section called “Obtención de notificaciones de implementación”.
ListDeployments	Devuelve el historial de implementaciones del grupo.
ResetDeployments	Restablece las implementaciones del grupo. Es posible que desee restablecer las implementaciones de grupo para mover o eliminar un grupo o para eliminar la informaci ón de implementación. Para obtener más

Acción de	Descripción
	información, consulte the section called "Restablecimiento de implementaciones".

> **Note**
>
> Para obtener información acerca de las operaciones de implementación por lotes, consulte the section called "Creación de implementaciones por lotes".

Obtener el ID del grupo

El ID del grupo se usa comúnmente en acciones de la API. Puede utilizar la ListGroupsacción para buscar el ID del grupo objetivo en su lista de grupos. Por ejemplo, en la AWS CLI, utilice el comando `list-groups`.

```
aws greengrass list-groups
```

También puede incluir la opción `query` para filtrar los resultados. Por ejemplo:

- Para obtener el grupo creado más recientemente:

  ```
  aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))[0]"
  ```

- Para obtener un grupo por nombre:

  ```
  aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
  ```

  No es necesario que los nombres de grupo sean únicos, por lo que podrían devolverse varios grupos.

A continuación se muestra un ejemplo de respuesta de `list-groups`. La información de cada grupo incluye el ID del grupo (en la propiedad `Id`) y el ID de la versión de grupo más reciente (en la propiedad `LatestVersion`). Para obtener otros ID de versión para un grupo, usa el ID de grupo con ListGroupVersions.

> **Note**
>
> También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

Si no especifica una Región de AWS, los comandos de la AWS CLI utilizan la región predeterminada de su perfil. Para devolver grupos en una región distinta, incluya la opción de *región*. Por ejemplo:

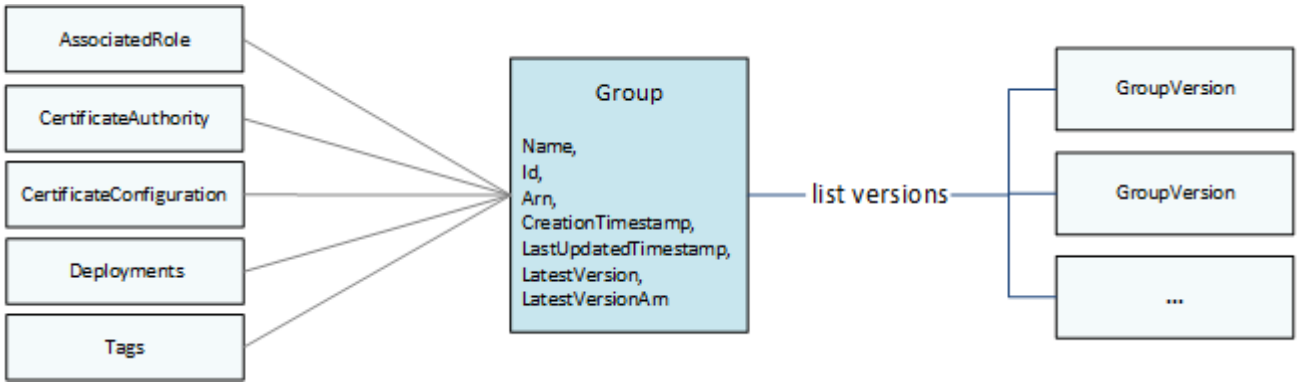
```
aws greengrass list-groups --region us-east-1
```

Información general sobre el modelo de objetos de grupo de AWS IoT Greengrass

Cuando se programa con la API de AWS IoT Greengrass, resulta útil conocer el modelo de objetos de grupo de Greengrass.

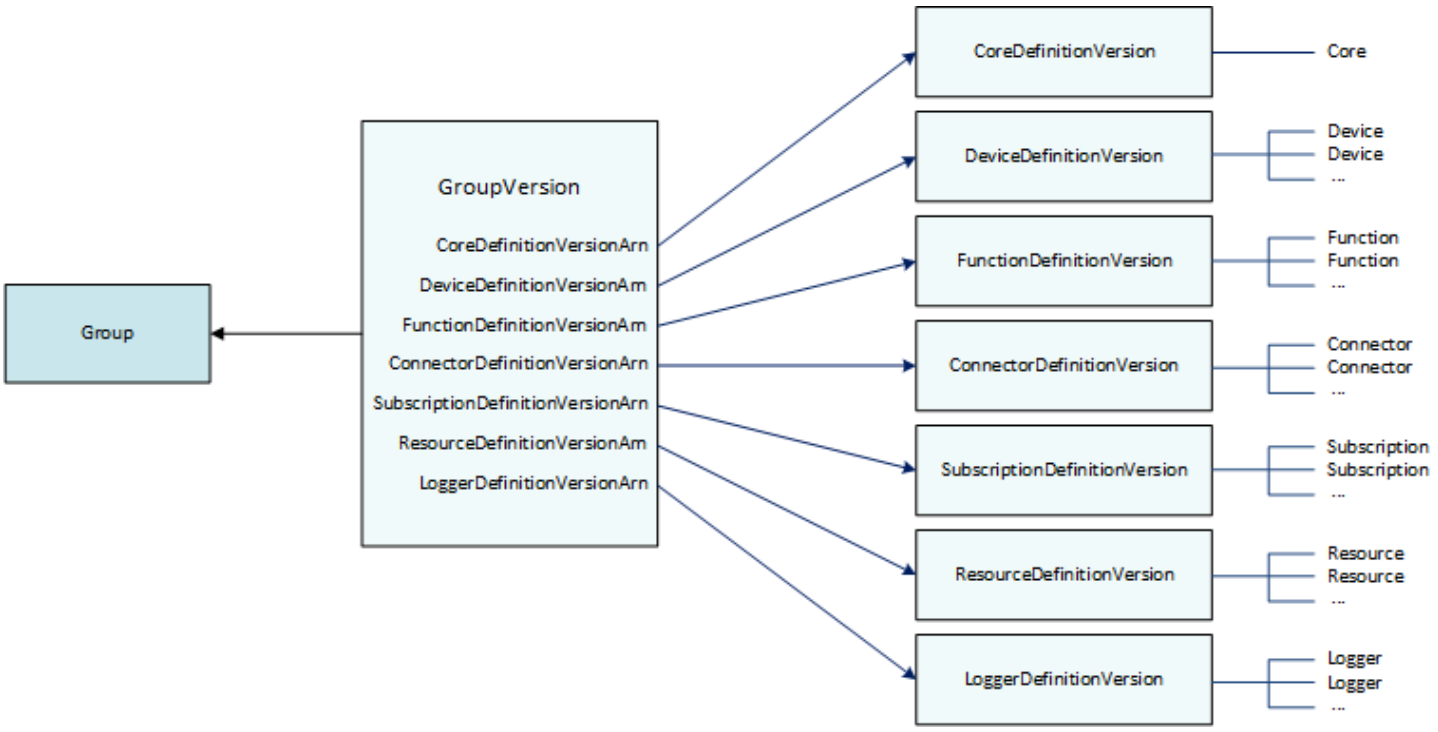
Grupos

En la API de AWS IoT Greengrass, el objeto `Group` de nivel superior se compone de metadatos y de una lista de objetos `GroupVersion`. Los objetos `GroupVersion` están asociados con un objeto `Group` a través de su ID.



Versiones del grupo

Los objetos `GroupVersion` definen la pertenencia a los grupos. Cada `GroupVersion` hace referencia a un objeto `CoreDefinitionVersion` y a otras versiones de componentes a través de su ARN. Estas referencias determinan qué entidades se van a incluir en el grupo.



Por ejemplo, para incluir tres funciones de Lambda, un dispositivo y dos suscripciones en el grupo, `GroupVersion` hace referencia a:

- El `CoreDefinitionVersion` que contiene el núcleo requerido.
- `FunctionDefinitionVersion` que contiene las tres funciones.
- La `DeviceDefinitionVersion` que contiene el dispositivo de cliente.
- `SubscriptionDefinitionVersion` que contiene las dos suscripciones.

El objeto `GroupVersion` implementado en un dispositivo de núcleo determina las entidades que están disponibles en el entorno local y cómo pueden interactuar.

Componentes del grupo

Los componentes que añada a los grupos tienen una jerarquía de tres niveles:

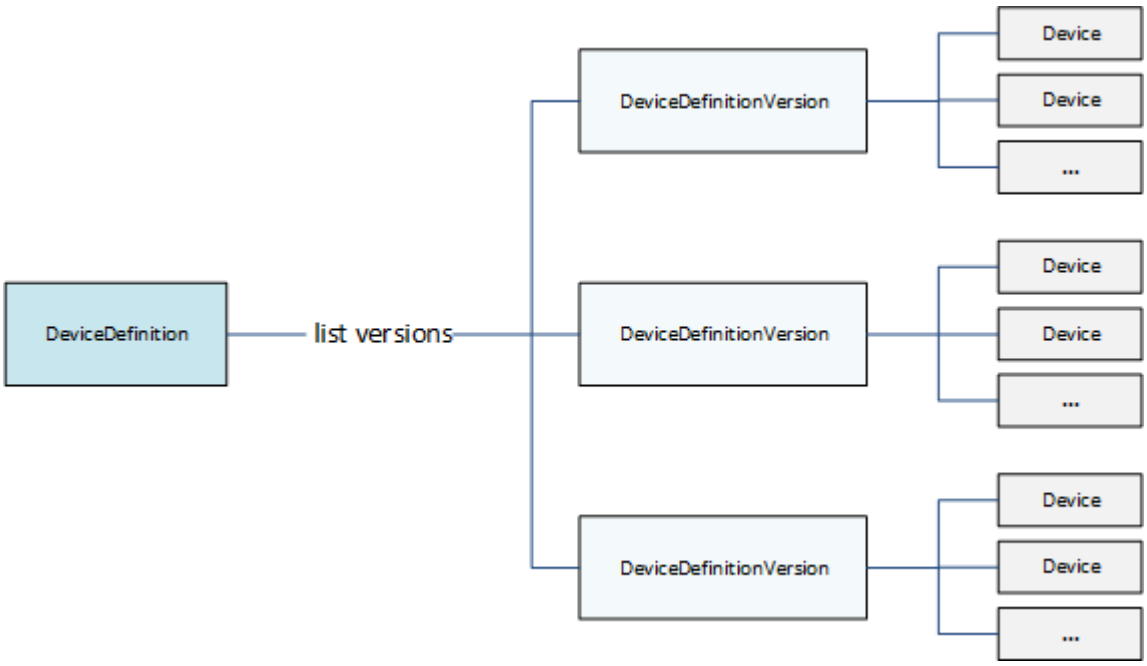
- Una definición que hace referencia a una lista de DefinitionVersionobjetos de un tipo determinado. Por ejemplo, un objeto `DeviceDefinition` hace referencia a una lista de objetos `DeviceDefinitionVersion`.
- A DefinitionVersionque contiene un conjunto de entidades de un tipo determinado. Por ejemplo, un objeto `DeviceDefinitionVersion` contiene una lista de objetos `Device`.

- Entidades individuales que definen sus propiedades y su comportamiento. Por ejemplo, un objeto `Device` define el ARN del dispositivo de cliente correspondiente en el registro de AWS IoT, el ARN de su certificado de dispositivo y si su instantánea local se sincroniza automáticamente con la nube.

  Puede añadir los siguientes tipos de entidades a un grupo:

  - Kinesis - S3
  - Core
  - Dispositivo
  - Función
  - Logger
  - Resource
  - Suscripción

En el siguiente ejemplo, `DeviceDefinition` hace referencia a tres objetos `DeviceDefinitionVersion`, cada uno de los cuales contiene varios objetos `Device`. En los grupos, solo se utiliza un `DeviceDefinitionVersion` cada vez.



Actualización de grupos

En la API de AWS IoT Greengrass, se utilizan versiones para actualizar la configuración de un grupo. Las versiones son inmutables, por lo que para añadir, eliminar o cambiar los componentes del grupo, debe crear DefinitionVersionobjetos que contengan entidades nuevas o actualizadas.

Puede asociar DefinitionVersionsobjetos nuevos a objetos de definición nuevos o existentes. Por ejemplo, puede utilizar la acción `CreateFunctionDefinition` para crear un objeto `FunctionDefinition` que incluya el objeto `FunctionDefinitionVersion` como versión inicial, o puede utilizar la acción `CreateFunctionDefinitionVersion` y hacer referencia a un objeto `FunctionDefinition` existente.

Tras crear los componentes del grupo, se crea uno `GroupVersion` que contenga todos los DefinitionVersionobjetos que se desean incluir en el grupo. A continuación, debe implementar `GroupVersion`.

Para implementar un objeto `GroupVersion`, debe hacer referencia a un objeto `CoreDefinitionVersion` que contenga exactamente un objeto `Core`. Todas las entidades a las que se haga referencia deben ser miembros del grupo. Además, debe haber un rol de servicio de Greengrass asociado a su Cuenta de AWS en la Región de AWS en la que va a implementar el `GroupVersion`.

> **Note**
>
> Las acciones `Update` de la API se utilizan para cambiar el nombre de un objeto `Group` o Definition del componente.

Actualización de entidades que hacen referencia a recursos de AWS

Las funciones de Lambda de Greengrass y los recursos secretos definen propiedades específicas de Greengrass y también hacen referencia a los recursos de AWS correspondientes. Para actualizar estas entidades, puede realizar cambios en el recurso de AWS correspondiente en lugar de en los objetos de Greengrass. Por ejemplo, las funciones DE Lambda hacen referencia a una función de AWS Lambda y también definen el ciclo de vida y otras propiedades específicas del grupo de Greengrass.

- Para actualizar el código de la función de Lambda o las dependencias empaquetadas, realice los cambios en AWS Lambda. Durante la siguiente implementación del grupo, estos cambios se recuperarán de AWS Lambda y se copiarán en él entorno local.
- Para actualizar las propiedades específicas de Greengrass, debe crear un objeto `FunctionDefinitionVersion` que contenga las propiedades `Function` actualizadas.

> **Note**
>
> Las funciones de Lambda de Greengrass pueden hacer referencia a una función de Lambda por el ARN del alias o el ARN de versión. Si hace referencia al ARN del alias (recomendado), no es necesario actualizar `FunctionDefinitionVersion` (ni `SubscriptionDefinitionVersion`) al publicar una nueva versión de la función en AWS Lambda. Para obtener más información, consulte the section called “Referencia a funciones por alias o versión”.

Véase también

- the section called “Obtención de notificaciones de implementación”
- the section called “Restablecimiento de implementaciones”
- the section called “Creación de implementaciones por lotes”
- Solución de problemas de implementación

- Referencia de la API de AWS IoT Greengrass Version 1
- Comandos de AWS IoT Greengrass en la Referencia de comandos de AWS CLI

Obtención de notificaciones de implementación

Las reglas de eventos de Amazon EventBridge le proporcionan notificaciones sobre los cambios de estado de sus implementaciones de grupos de Greengrass. EventBridge proporciona una secuencia de eventos de sistema casi en tiempo real que describe cambios en los recursos de AWS. AWS IoT Greengrass envía estos eventos a EventBridge siguiendo el criterio de al menos una vez. Esto significa que AWS IoT Greengrass puede enviar varias copias de un evento determinado para

garantizar la entrega. Además, los agentes de escucha de eventos podrían no recibir los eventos en el orden en el que los eventos han ocurrido.

> **Note**
>
> Amazon EventBridge es un servicio de bus de eventos que puede utilizar para conectar sus aplicaciones con datos procedentes de diversas fuentes, como los dispositivos centrales de Greengrass y las notificaciones de implementación. Para obtener más información, consulte What is Amazon EventBridge? (¿Qué es Amazon EventBridge?) en la Guía del usuario de Amazon EventBridge.

AWS IoT Greengrass emite un evento cuando las implementaciones de grupo cambian de estado. Puede crear una regla de EventBridge que se ejecute en todas las transiciones de estado que especifique. Cuando una implementación entra en un estado que inicia una regla, EventBridge invoca las acciones de destino definidas en la regla. Esto le permite enviar notificaciones, capturar información sobre el evento, tomar medidas correctivas o iniciar otros eventos en respuesta a un cambio de estado. Por ejemplo, puede crear reglas para los siguientes casos de uso:

- Iniciar operaciones posteriores a la implementación, como descargar recursos y enviar notificaciones al personal.
- Envíe notificaciones en caso de una implementación correcta o con error.
- Publicar métricas personalizadas sobre los eventos de implementación.

AWS IoT Greengrass emite un evento cuando una implementación adopta los siguientes estados: `Building`, `InProgress`, `Success` y `Failure`.

> **Note**
>
> Actualmente, no se puede monitorizar el estado de una operación de implementación por lotes . Sin embargo, AWS IoT Greengrass emite eventos de cambio de estado sobre las implementaciones de los grupos específicos que forman parte de una implementación por lotes.

Evento de cambio de estado de una implementación de grupo

El evento de un cambio de estado de la implementación tiene el siguiente formato:

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "time":"2018-03-22T00:38:11Z",
    "region":"us-west-2",
    "resources":[],
    "detail":{
        "group-id": "284dcd4e-24bc-4c8c-a770-EXAMPLEf03b8",
        "deployment-id": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "deployment-type": "NewDeployment|Redeployment|ResetDeployment|
ForceResetDeployment",
        "status": "Building|InProgress|Success|Failure"
    }
}
```

Puede crear reglas que se apliquen a uno o varios grupos. Puede filtrar reglas por uno o varios de los siguientes tipos de implementaciones y estados de implementación:

Tipos de implementación

- `NewDeployment`. Primera implementación de la versión de un grupo.
- `ReDeployment`. Nueva implementación de la versión de un grupo.
- `ResetDeployment`. Elimina la información de implementación almacenada en la Nube de AWS y en el núcleo de AWS IoT Greengrass. Para obtener más información, consulte the section called “Restablecimiento de implementaciones”.
- `ForceResetDeployment`. Elimina la información de la implementación almacenada en la Nube de AWS y notifica el éxito sin esperar a que el núcleo responda. También elimina la información de la implementación almacenada en el núcleo si este está conectado. De lo contrario, la información se eliminará la próxima vez que se conecte el núcleo.

Estados de implementación

- `Building`. AWS IoT Greengrass está validando la configuración del grupo y creando artefactos de implementación.

- `InProgress`. La implementación está en curso en AWS IoT Greengrass.
- `Success`. La implementación se ha realizado correctamente.
- `Failure`. La implementación no se ha realizado correctamente

Es posible que los eventos se dupliquen o estén desordenados. Para determinar el orden de los eventos, utilice la propiedad `time`.

> **Note**
>
> AWS IoT Greengrass no utiliza la propiedad `resources`, por lo que siempre está vacía.

Prerequisitos para crear las reglas de EventBridge

Antes de crear una regla de EventBridge para AWS IoT Greengrass, haga lo siguiente:

- Familiarizase con los eventos, las reglas y los destinos de EventBridge.
- Cree y configure los destinos que las reglas de EventBridge han invocado. Las reglas pueden invocar muchos tipos de destinos, entre los que se incluyen:
  - Amazon Simple Notification Service (Amazon SNS)
  - Funciones de AWS Lambda
  - Amazon Kinesis Video Streams
  - Colas de Amazon Simple Queue Service (Amazon SQS)

Para obtener más información, consulte What is Amazon EventBridge? (¿Qué es Amazon EventBridge?) y Getting started with Amazon EventBridge (Introducción a Amazon EventBridge) en la Guía del usuario de Amazon EventBridge.

Configuración de las notificaciones de implementación (consola)

Siga estos pasos para crear una regla de EventBridge que publique un tema de Amazon SNS cuando cambie el estado de implementación de un grupo. De este modo, los servidores web, las direcciones de correo electrónico y otros suscriptores del tema podrán responder al evento. Para obtener más información, consulte Creación de una regla de EventBridge que se active a partir de un evento de un recurso de AWS en la Guía del usuario de Amazon EventBridge.

1. Abra la consola de Amazon EventBridge.
2. En el panel de navegación, seleccione Rules (Reglas).
3. Elija Create rule.
4. Escriba un nombre y una descripción de la regla.

   Una regla no puede tener el mismo nombre que otra regla de la misma región y del mismo bus de eventos.
5. En Event bus (Bus de eventos), elija el bus de eventos que desea asociar a esta regla. Si desea que esta regla coincida con eventos procedentes de su cuenta, seleccione Bus de eventos predeterminado de AWS. Cuando un servicio de AWS en su cuenta emite un evento, siempre va al bus de eventos predeterminado de su cuenta.
6. En Rule type (Tipo de regla), elija Rule with an event pattern (Regla con un patrón de evento).
7. Elija Next (Siguiente).
8. En Event source (Origen del evento), elija AWS services (Servicios de).
9. En Patrón de eventos, seleccione serviciosAWS.
10. En AWSServicio, elija Greengrass.
11. En Event type (Tipo de evento), elija Greengrass Deployment Status Change (Cambio de estado de implementación de Greengrass).

    > **Note**
    >
    > El tipo de evento Llamada a la API de AWS mediante CloudTrail se basa en la integración de AWS IoT Greengrass con AWS CloudTrail. Puede usar esta opción para crear reglas iniciadas por llamadas de lectura o escritura a la API de AWS IoT Greengrass. Para obtener más información, consulte the section called “Registro de llamadas a la API de AWS IoT Greengrass con AWS CloudTrail”.
12. Elija los estados de implementación que inicia una notificación.

    - Para recibir notificaciones de todos los eventos de cambio de estado, seleccione Any state (Cualquier estado).
    - Para recibir notificaciones solo para algunos eventos de cambio de estado, elija Specific state(s) (Estados específicos) y, a continuación, elija los estados de destino.
13. Elija los tipos de implementación que inicia una notificación.

- Para recibir notificaciones de todos los tipos de implementación, elija Any state (Cualquier estado).
- Para recibir notificaciones solo para algunos tipos de implementación, elija Specific state(s) (Estados específicos) y, a continuación, elija los tipos de implementación de destino.

14. Elija Next (Siguiente).
15. En Tipos de destino (Tipos de destino), elija AWS service.
16. En Seleccionar destinos, configure su destino. En este ejemplo se utiliza un tema de Amazon SNS, pero se pueden configurar otros tipos de destino para enviar notificaciones.

    a. En Target (Destino), elija SNS topic (Tema de SNS).
    b. En Topic (Tema), elija el tema de destino.
    c. Elija Next (Siguiente).
17. En Etiquetas, defina etiquetas para la regla o deje los campos vacíos.
18. Elija Next (Siguiente).
19. Revise los detalles de la regla y elija Create rule (Crear regla).

Configuración de notificaciones de implementación (CLI)

Siga estos pasos para crear una regla de EventBridge que publique un tema de Amazon SNS cuando cambie el estado de implementación de un grupo. De este modo, los servidores web, las direcciones de correo electrónico y otros suscriptores del tema podrán responder al evento.

1. Cree la regla de .

   - Sustituya *id-grupo* por el ID de su grupo de AWS IoT Greengrass.

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"group-id\":
    [\"group-id\"]}}"
   ```

   Las propiedades que se omiten en el patrón no se tienen en cuenta.
2. Añada el tema como destino de la regla.

   - Sustituya *topic-arn* por el ARN de su tema de Amazon SNS.

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="topic-arn"
```

> **Note**
>
> Para permitir que Amazon EventBridge llame al tema de destino, debe agregar en el tema una política basada en recursos. Para obtener más información, consulte Amazon SNS permissions (Permisos de Amazon SNS) en la Guía del usuario de Amazon EventBridge.

Para obtener más información, consulte Events and event patterns in EventBridge (Eventos y patrones de eventos en EventBridge) en la Guía del usuario de Amazon EventBridge.

Configuración de notificaciones de implementación (AWS CloudFormation)

Utilice plantillas de AWS CloudFormation para crear reglas de EventBridge que envíen notificaciones sobre cambios de estado para las implementaciones de grupos de Greengrass. Para obtener más información, consulte Referencia de tipos de recursos de Amazon EventBridge en la Guía del usuario de AWS CloudFormation.

Véase también

- Implementación de grupos de AWS IoT Greengrass
- ¿Qué es Amazon EventBridge? en la Guía del usuario de Amazon EventBridge

Restablecimiento de implementaciones

Esta característica está disponible para AWS IoT Greengrass Core versión 1.1 y posteriores.

Es posible que desee restablecer las implementaciones de un grupo para:

- Elimine el grupo, por ejemplo, cuando desee mover el núcleo del grupo a otro grupo o cuando se haya rediseñado el núcleo del grupo. Antes de eliminar un grupo, debe restablecer las implementaciones del grupo para usar el núcleo con otro grupo de Greengrass.

- Mover el núcleo del grupo a otro grupo
- Revertir el grupo al estado que tenía antes de la implementación.
- Eliminar la configuración de la implementación del dispositivo del núcleo
- Eliminar datos confidenciales del dispositivo del núcleo o de la nube
- Implementar una nueva configuración del grupo en un núcleo sin tener que reemplazar el núcleo por otro del grupo actual

> **Note**
>
> La funcionalidad de restablecimiento de implementaciones no está disponible en AWS IoT Greengrass Core Software v1.0.0. No puede eliminar los grupos implementados con la versión 1.0.0.

La operación de restablecimiento primero limpia toda la información que hay en la nube sobre la implementación de un determinado grupo. A continuación, ordena al dispositivo del núcleo del grupo que borre toda la información relacionada con la implementación (funciones de Lambda, registros de usuario, base de datos de instantáneas y certificado del servidor, aunque no el archivo `config.json` definido por el usuario ni los certificados del núcleo de Greengrass). No se puede iniciar el restablecimiento de la implementación de un grupo si actualmente dicho grupo tiene una implementación con el estado `In Progress` o `Building`.

Restablecimiento de implementaciones desde la consola de AWS IoT

Puede restablecer una implementación de grupo desde la página de configuración del grupo de la consola de AWS IoT.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. En la pestaña Implementaciones, elija Reiniciar implementación.
4. En el cuadro de diálogo Restablecer implementaciones para este grupo de Greengrass, escriba **`confirm`** para aceptar y seleccione Restablecer implementación.

Restablecimiento de implementaciones con la API de AWS IoT Greengrass

Puede utilizar la acción `ResetDeployments` de la AWS CLI, la API de AWS IoT Greengrass o el SDK de AWS para restablecer las implementaciones. En los ejemplos de este tema, se utiliza la CLI.

```
aws greengrass reset-deployments --group-id GroupId [--force]
```

Argumentos del comando de la CLI **`reset-deployments`**:

`--group-id`

El ID del grupo. Utilice el comando `list-groups` para obtener este valor.

`--force`

Opcional. Utilice este parámetro si el dispositivo del núcleo del grupo se ha perdido, robado o destruido. Esta opción hace que el proceso de restablecimiento de la implementación notifique que se ha realizado correctamente una vez que se ha borrado toda la información de implementación de la nube, sin tener que esperar a que responda el dispositivo del núcleo. Sin embargo, si el dispositivo del núcleo está activo o se activa, también realiza operaciones de limpieza.

El resultado del comando `reset-deployments` de la CLI es similar al siguiente:

```
{
    "DeploymentId": "4db95ef8-9309-4774-95a4-eea580b6ceef",
    "DeploymentArn": "arn:aws:greengrass:us-west-2:106511594199:/greengrass/groups/
b744ed45-a7df-4227-860a-8d4492caa412/deployments/4db95ef8-9309-4774-95a4-eea580b6ceef"
}
```

Puede comprobar el estado del restablecimiento de la implementación con el comando de la CLI `get-deployment-status`:

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

Argumentos del comando de la CLI **`get-deployment-status`**:

`--deployment-id`

El ID de implementación.

`--group-id`

El ID del grupo.

El resultado del comando `get-deployment-status` de la CLI es similar al siguiente:

```
{
    "DeploymentStatus": "Success",
    "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

El estado de `DeploymentStatus` se establece en `Building` cuando se está preparando el restablecimiento de la implementación. Cuando el restablecimiento de la implementación está listo, pero el AWS IoT Greengrass principal aún no lo ha aplicado, el estado de `DeploymentStatus` es `InProgress`.

Si la operación de restablecimiento da un error, se devuelve la información del error en la solicitud.

Véase también

- Implementación de grupos de AWS IoT Greengrass
- ResetDeployments en la referencia AWS IoT Greengrass Version 1 de la API
- GetDeploymentStatusen la referencia AWS IoT Greengrass Version 1 de la API

Creación de implementaciones de grupos por lotes

Puede utilizar llamadas a la API simples para implementar un gran número de grupos de Greengrass a la vez. Estas implementaciones se activan con una velocidad flexible que tiene un límite superior fijo.

En este tutorial se describe cómo utilizar la AWS CLI para crear y monitorizar un grupo de implementación por lotes en AWS IoT Greengrass. El ejemplo de implementación por lotes de este tutorial contiene varios grupos. Puede utilizar el ejemplo en su implementación para añadir tantos grupos como necesite.

El tutorial contiene los siguientes pasos generales:

1. Crear y cargar el archivo de entrada de la implementación por lotes

2. Cree y configure un rol de ejecución IAM para implementaciones masivas
3. Permitir que el rol de ejecución acceda al bucket de S3
4. Implementar los grupos
5. Probar la implementación

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Uno o varios grupos de Greengrass implementables. Para obtener más información acerca de la creación de núcleos y grupos de AWS IoT Greengrass, consulte Empezar con AWS IoT Greengrass.
- La AWS CLI instalada y configurada en la máquina. Para obtener más información, consulte Guía del usuario de AWS CLI.
- Un bucket de S3 creado en la misma Región de AWS que AWS IoT Greengrass. Para obtener más información, consulte Creación y configuración de un bucket de S3 en la Guía del usuario del Servicio de almacenamiento sencillo de Amazon.

> **Note**
>
> Actualmente, los buckets habilitados para SSE KMS no son compatibles.

Paso 1: Crear y cargar el archivo de entrada de la implementación por lotes

En este paso, creará un archivo de entrada de implementación y lo cargará en el bucket de Amazon S3. Este archivo es un archivo JSON delimitado por líneas y serializado que contiene información acerca de cada grupo de la implementación por lotes. AWS IoT Greengrass utiliza esta información para implementar cada grupo en su nombre cuando inicie la implementación de grupos por lotes.

1. Ejecute el siguiente comando con el fin de obtener el parámetro `groupId` para cada grupo que desea implementar. Escriba el parámetro `groupId` en el archivo de entrada de implementación por lotes, de modo que AWS IoT Greengrass pueda identificar cada grupo que se va a implementar.

> **Note**
>
> También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

```
aws greengrass list-groups
```

La respuesta contiene información acerca de cada grupo de su cuenta de AWS IoT Greengrass:

```
{
  "Groups": [
    {
      "Name": "string",
      "Id": "string",
      "Arn": "string",
      "LastUpdatedTimestamp": "string",
      "CreationTimestamp": "string",
      "LatestVersion": "string",
      "LatestVersionArn": "string"
    }
  ],
  "NextToken": "string"
}
```

Ejecute el siguiente comando con el fin de obtener el parámetro `groupVersionId` de cada grupo que desea implementar.

```
list-group-versions --group-id groupId
```

La respuesta contiene información acerca de todas las versiones de grupo. Anote el valor de la `Version` del grupo que desee utilizar.

```
{
  "Versions": [
    {
      "Arn": "string",
      "Id": "string",
      "Version": "string",
      "CreationTimestamp": "string"
    }
  ],
  "NextToken": "string"
}
```

2. En la terminal del equipo o el editor que prefiera, cree un archivo, *MyBulkDeploymentInputFile*, desde el siguiente ejemplo. Este archivo contiene información acerca de cada grupo de AWS IoT Greengrass que se debe incluir en una implementación por lotes. Aunque este ejemplo define varios grupos, para este tutorial, el archivo puede contener solo uno.

> **Note**
>
> El tamaño de este archivo debe ser inferior a 100 MB.

```
{"GroupId":"groupId1", "GroupVersionId":"groupVersionId1",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId2", "GroupVersionId":"groupVersionId2",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId3", "GroupVersionId":"groupVersionId3",
 "DeploymentType":"NewDeployment"}
...
```

Cada registro (o línea) contiene un objeto de grupo. Cada objeto de grupo contiene su correspondiente `GroupId` y `GroupVersionId`, y un parámetro `DeploymentType`. Actualmente, AWS IoT Greengrass solo admite tipos de implementación por lotes de `NewDeployment`.

Guarde y cierre el archivo. Anote la ubicación del archivo.

3. Utilice el siguiente comando en el terminal para cargar el archivo de entrada en el bucket de Amazon S3. Sustituya la ruta del archivo por la ubicación y el nombre del archivo. Para obtener información, consulte Cargar un objeto en un bucket.

```
aws s3 cp path/MyBulkDeploymentInputFile s3://my-bucket/
```

Paso 2: Crear y configurar un rol de ejecución de IAM

En este paso, utilizará la consola de IAM para crear un rol de ejecución independiente. A continuación, establezca una relación de confianza entre el rol y AWS IoT Greengrass, y asegúrese de que el usuario de IAM tenga privilegios de `PassRole` para el rol de ejecución. Esto permite que AWS IoT Greengrass asuma el rol de ejecución y cree las implementaciones en su nombre.

1. Utilice la siguiente política para crear un rol de ejecución. Este documento de política permite a AWS IoT Greengrass acceder al archivo de entrada de implementación por lotes al crear cada implementación en su nombre.

   Para obtener más información acerca de cómo crear un rol de IAM y delegar permisos, consulte Creación de roles de IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": "VisualEditor0",
           "Effect": "Allow",
           "Action": "greengrass:CreateDeployment",
           "Resource": [
               "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId1",
               "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId2",
               "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId3",
               ...
           ]
       }
   ]
}
```

> **Note**
>
> Esta política debe tener un recurso para cada grupo o versión de grupo en el archivo de entrada de implementación por lotes que va a implementar AWS IoT Greengrass. Para permitir el acceso a todos los grupos, en `Resource`, especifique un asterisco:
>
> ```
> "Resource": ["*"]
> ```

2. Modifique la relación de confianza para que el rol de ejecución incluya AWS IoT Greengrass. Esto permite a AWS IoT Greengrass utilizar el rol de ejecución y los permisos asociados a él. Para obtener más información, consulte Edición de la relación de confianza para un rol existente.

   Le recomendamos que incluya también las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en su política de confianza para ayudar a prevenir el problema de seguridad del suplente confuso. Las claves de contexto de condición restringen el acceso para permitir solo las solicitudes que provienen de la cuenta especificada y del espacio de trabajo de Greengrass. Para obtener más información sobre el problema del suplente confuso, consulte Prevención del suplente confuso entre servicios.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
```

```
    }
  ]
}
```

3. Conceda a IAM los permisos `PassRole` del rol de ejecución al usuario de IAM. Este usuario de IAM es el que se utiliza para iniciar la implementación por lotes. Los permisos `PassRole` permiten su usuario de IAM transferir el rol de ejecución para AWS IoT Greengrass lo use. Para obtener más información, consulte Concesión de permisos a un usuario para transferir un rol a un servicio de AWS.

   Utilice el siguiente ejemplo para actualizar la política de IAM adjunta a su rol de ejecución. Modifique este ejemplo según sea necesario.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": "Stmt1508193814000",
           "Effect": "Allow",
           "Action": [
               "iam:PassRole"
           ],
           "Resource": [
               "arn:aws:iam::account-id:user/executionRoleArn"
           ]
           "Condition": {
               "StringEquals": {
                   "iam:PassedToService": "greengrass.amazonaws.com"
               }
           }
       }
   ]
}
```

Paso 3: Permitir que el rol de ejecución acceda al bucket de S3

Para iniciar la implementación por lotes, el rol de ejecución debe poder leer el archivo de entrada de implementación por lotes desde el bucket de Amazon S3. Asocie la siguiente política de ejemplo su bucket de Amazon S3, por lo que el rol de ejecución puede acceder a los permisos `GetObject`.

Para obtener más información, consulte ¿Cómo agrego una política de bucket en S3?

```
{
    "Version": "2008-10-17",
    "Id": "examplePolicy",
    "Statement": [
        {
            "Sid": "Stmt1535408982966",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "executionRoleArn"
                ]
            },
            "Action": "s3:GetObject",
            "Resource":
            "arn:aws:s3:::my-bucket/objectKey"
        }
    ]
}
```

Puede utilizar el siguiente comando en el terminal para comprobar la política del bucket.

```
aws s3api get-bucket-policy --bucket my-bucket
```

Note

Puede modificar directamente el rol de ejecución para concederle el permiso `GetObject` al bucket de Amazon S3. Para hacerlo, asocie la siguiente política de ejemplo al rol de ejecución.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-bucket/objectKey"
```

```
        }
    ]
}
```

Paso 4: Implementar los grupos

En este paso, iniciará una operación de implementación por lotes para todas las versiones de grupo configuradas en el archivo de entrada de implementación por lotes. La acción de implementación de cada una de las versiones de grupo es del tipo `NewDeploymentType`.

> **Note**
>
> No puede llamar a StartBulkDeployment si hay otra implementación por lotes de la misma cuenta ejecutándose. La solicitud se rechaza.

1. Utilice el siguiente comando para iniciar la implementación por lotes.

   Le recomendamos que incluya un token `X-Amzn-Client-Token` en cada solicitud StartBulkDeployment. Estas solicitudes son idempotentes en relación con el token y los parámetros de solicitud. Este token puede ser cualquier cadena única que distingue entre mayúsculas y minúsculas de hasta 64 caracteres ASCII.

   ```
   aws greengrass start-bulk-deployment --cli-input-json "{
             "InputFileUri":"URI of file in S3 bucket",
             "ExecutionRoleArn":"ARN of execution role",
             "AmznClientToken":"your Amazon client token"
             }"
   ```

   El comando debe generar un código de estado correcto de `200`, junto con la siguiente respuesta:

   ```
   {
     "bulkDeploymentId": UUID
   }
   ```

Anote el ID de la implementación por lotes. Se puede utilizar para comprobar el estado de la implementación por lotes.

> **Note**
>
> Aunque actualmente no se admiten las operaciones de implementación por lotes, puede crear reglas de eventos de Amazon EventBridge para recibir notificaciones sobre los cambios de estado de la implementación registrados por cada grupo. Para obtener más información, consulte the section called “Obtención de notificaciones de implementación”.

2. Use el siguiente comando para comprobar el estado de la implementación por lotes.

```
aws greengrass get-bulk-deployment-status --bulk-deployment-id 1234567
```

El comando debe devolver un código de estado de`200`, además de una carga JSON de información:

```
 {
  "BulkDeploymentStatus": Running,
  "Statistics": {
     "RecordsProcessed": integer,
     "InvalidInputRecords": integer,
     "RetryAttempts": integer
  },
  "CreatedAt": "string",
  "ErrorMessage": "string",
  "ErrorDetails": [
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
  ]
}
```

`BulkDeploymentStatus` contiene el estado actual de la ejecución por lotes. La ejecución puede tener uno de seis estados diferentes:

- `Initializing`. La solicitud de implementación por lotes se ha recibido y la ejecución se está preparando para iniciarse.
- `Running`. La ejecución de la implementación por lotes se ha iniciado.
- `Completed`. La ejecución de la implementación por lotes ha terminado de procesar todos los registros.
- `Stopping`. La ejecución de la implementación por lotes ha recibido un comando para detenerse y terminará en breve. No se puede iniciar una nueva implementación por lotes si hay una anterior con el estado `Stopping`.
- `Stopped`. La ejecución de la implementación por lotes se ha detenido manualmente.
- `Failed`. La ejecución de la implementación por lotes ha encontrado un error y ha terminado. Puede ver los detalles del error en el campo `ErrorDetails`.

La carga JSON también incluye información estadística acerca del progreso de la implementación por lotes. Puede utilizar esta información para determinar la cantidad de grupos que se han procesado y cuántos han obtenido un error. La información estadística incluye los siguientes datos:

- `RecordsProcessed`: el número de registros de grupo que se intentaron.
- `InvalidInputRecords`: el número total de registros que han devuelto un error que no se puede reintentar. Por ejemplo, esto puede ocurrir si un registro de grupo del archivo de entrada utiliza un formato no válido o especifica una versión de grupo que no existe, o bien si la ejecución no concede permiso para implementar un grupo o una versión de grupo.
- `RetryAttempts`: el número de intentos de implementación que han devuelto un error que se puede reintentar. Por ejemplo, un reintento se activa si el intento para implementar un grupo devuelve un error de limitación controlada. Una implementación de grupo puede reintentarse hasta cinco veces.

En el caso de que se produzca un error de una ejecución de implementación por lotes, esta carga también incluye una sección `ErrorDetails` que se puede utilizar para la resolución de problemas. Contiene información acerca de la causa del error de ejecución.

Puede comprobar periódicamente el estado de la implementación por lotes para confirmar que está avanzando según lo previsto. Una vez finalizada la implementación, `RecordsProcessed` debe ser igual que el número de grupos de implementación del archivo de entrada de implementación por lotes. Esto indica que cada registro se ha procesado.

Paso 5: Probar la implementación

Utilice el comando ListBulkDeployments para encontrar el ID de la implementación por lotes.

```
aws greengrass list-bulk-deployments
```

Este comando devuelve una lista de todas las implementaciones por lote de la más reciente a la que menos, incluido el parámetro `BulkDeploymentId`.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": 1234567,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

Ahora llame al comando ListBulkDeploymentDetailedReports para recopilar información detallada acerca de cada implementación.

```
aws greengrass list-bulk-deployment-detailed-reports --bulk-deployment-id 1234567
```

El comando debe devolver un código de estado de`200`, además de una carga JSON de información:

```
{
  "BulkDeploymentResults": [
    {
      "DeploymentId": "string",
      "GroupVersionedArn": "string",
      "CreatedAt": "string",
      "DeploymentStatus": "string",
      "ErrorMessage": "string",
      "ErrorDetails": [
        {
          "DetailedErrorCode": "string",
          "DetailedErrorMessage": "string"
        }
      ]
    }
  ],
  "NextToken": "string"
}
```

Esta carga normalmente contiene una lista paginada de cada implementación con su estado de la más reciente a la que menos. También contiene más información en caso de que se produzca un error de ejecución de implementación por lotes. Una vez más, el número total de las implementaciones disponibles debe ser igual que el número de grupos que haya identificado en el archivo de entrada de implementación por lotes.

La información que se devuelve puede cambiar hasta que las implementaciones se encuentran en un estado terminal (éxito o error). Puede llamar a este comando periódicamente hasta entonces.

Solución de problemas de las implementaciones por lotes

Si la implementación por lotes no se realiza correctamente, puede seguir estos pasos de solución de problemas. Ejecute el comando en el terminal.

Solución de problemas del archivo de entrada

La implementación por lotes puede obtener un error en caso de que se produzca errores de sintaxis en el archivo de entrada de implementación por lotes. Esto devuelve un estado de implementación por lotes de `Failed` con un mensaje de error en el que se indica el número de línea del primer error de validación. Existen cuatro posibles errores:

- `InvalidInputFile: Missing GroupId at line number: line number`

Este error indica que la línea del archivo de entrada determinada no puede registrar el parámetro especificado. Los posibles parámetros que faltan son `GroupId` y `GroupVersionId`.

- ```
  InvalidInputFile: Invalid deployment type at line number : line number. Only valid
   type is 'NewDeployment'.
  ```

  Este error indica que la línea del archivo de entrada determinada muestra un tipo de implementación no válido. En este momento, el único tipo de implementación admitido es `NewDeployment`.

- ```
  Line %s is too long in S3 File. Valid line is less than 256 chars.
  ```

  Este error indica que la línea del archivo de entrada determinada es demasiado largo y debe acortarse.

- ```
  Failed to parse input file at line number: line number
  ```

  Este error indica que la línea del archivo de entrada determinada no se considera un JSON válido.

Comprobación de si hay implementaciones por lotes simultáneas

No puede iniciar una nueva implementación por lotes si hay otra ejecutándose o con un estado no terminal. Esto puede generar un error `Concurrent Deployment Error`. Puede utilizar el comando ListBulkDeployments para verificar que actualmente no hay ninguna implementación por lotes. Este comando muestra las implementaciones por lotes de la más reciente a la que menos.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": BulkDeploymentId,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
```
```

```
}
```

Utilice el parámetro `BulkDeploymentId` de la primera implementación por lotes enumerada para ejecutar el comando GetBulkDeploymentStatus. Si la implementación por lotes más reciente se encuentra en un estado de ejecución (`Initializing` o `Running`), utilice el siguiente comando para detener la implementación por lotes.

```
aws greengrass stop-bulk-deployment --bulk-deployment-id BulkDeploymentId
```

Esta acción genera un estado de `Stopping` hasta que la implementación obtiene el estado de `Stopped`. Después de que la implementación haya alcanzado el estado de `Stopped`, puede iniciar una nueva implementación por lotes.

Comprobación de ErrorDetails

Ejecute el comando `GetBulkDeploymentStatus` para devolver una carga JSON con información detallada sobre cualquier error de ejecución de la implementación por lotes.

```
"Message": "string",
"ErrorDetails": [
  {
    "DetailedErrorCode": "string",
    "DetailedErrorMessage": "string"
  }
]
```

Al salir con un error, la carga JSON `ErrorDetails` que devuelve esta llamada contiene obtener más información acerca del error de ejecución de implementación por lotes. Un código de estado de error de la serie `400`, por ejemplo, indica un error de entrada, ya sea en los parámetros de entrada o en las dependencias del intermediario.

Consultar el registro básico de AWS IoT Greengrass

Puede solucionar los problemas visualizando los registros de AWS IoT Greengrass Core. Use los siguientes comandos para ver `runtime.log`:

```
cd /greengrass/ggc/var/log
sudo cat system/runtime.log | more
```

Para obtener más información acerca del registro en AWS IoT Greengrass, consulte Monitorización con registros de AWS IoT Greengrass.

Véase también

Para obtener más información, consulte los siguientes recursos:

- Implementación de grupos de AWS IoT Greengrass
- :Los comandos de la API S3 de Amazon en la Referencia de comandos de la AWS CLI
- Comandos de AWS IoT Greengrass en la Referencia de comandos de AWS CLI

Ejecutar funciones de Lambda en el núcleo de AWS IoT Greengrass

AWS IoT Greengrass ofrece un entorno de tiempo de ejecución de Lambda en contenedores para código definido por el usuario que usted autoriza en AWS Lambda. Las funciones de Lambda que se implementa en un núcleo de AWS IoT Greengrass se ejecutan en el tiempo de ejecución de Lambda local del núcleo. Las funciones de Lambda locales pueden ser activadas por eventos locales, mensajes de la nube y otros orígenes, lo que lleva la funcionalidad de computación local a los dispositivos del cliente. Por ejemplo, puede utilizar funciones de Lambda Greengrass para filtrar los datos del dispositivo antes de transmitir los datos a la nube.

Para implementar una función de Lambda en un núcleo, añada la función a un grupo de Greengrass (haciendo referencia a la función de Lambda existente), defina una configuración específica del grupo y, a continuación, implemente el grupo. Si la función obtiene acceso a servicios de AWS, también debe agregar los permisos necesarios para el rol del grupo de Greengrass.

Puede configurar parámetros que determinan cómo se ejecutan las funciones de Lambda, incluidos permisos, aislamiento, límites de memoria, etc. Para obtener más información, consulte the section called “Control de la ejecución de la función de Lambda de Greengrass”.

> **Note**
>
> Esta configuración también permite ejecutar AWS IoT Greengrass en un contenedor de Docker. Para obtener más información, consulte the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”.

En la siguiente tabla se muestran los entornos de tiempo de ejecución de AWS Lambda compatibles y las versiones del software de AWS IoT Greengrass Core en las que puedan ejecutarse.

Lenguaje o plataforma	Versión de GGC
Python 3.8	1.11
Python 3.7	1.9 o posteriores
Python 2.7 *	1.0 o posteriores

Lenguaje o plataforma	Versión de GGC
Java 8	1.1 o posteriores
Node.js 12.x *	1.10 o posteriores
Node.js 8.10 *	1.9 o posteriores
Node.js 6.10 *	1.1 o posteriores
C, C++	1.6 o posteriores

* Puede ejecutar funciones de Lambda que utilicen estos entornos de tiempos de ejecución en versiones compatibles de AWS IoT Greengrass, pero no puede crearlas en AWS Lambda. Si el tiempo de ejecución de su dispositivo es diferente del tiempo de ejecución de Lambda AWS especificado para esa función, puede elegir su propio tiempo de ejecución utilizando `FunctionRuntimeOverride` en `FunctionDefintionVersion`. Para obtener más información, consulte CreateFunctionDefinition. Para más informació sobre los tiempos de ejecución compatibles, consulte la Política de soporte en tiempo de ejecución en la Guía de desarrolladores de AWS Lambda.

SDK para funciones de Lambda de Greengrass

AWS ofrece tres SDK que pueden utilizar las funciones de Lambda de Greengrass que se ejecutan en un núcleo de AWS IoT Greengrass. Estos SDK están contenidos en distintos paquetes, por lo que las funciones pueden utilizarlos de forma simultánea. Para utilizar un SDK en una función de Lambda de Greengrass, inclúyalo en el paquete de implementación de funciones de Lambda que subió a AWS Lambda.

SDK de AWS IoT Greengrass Core

Permite que las funciones de Lambda locales interactúen con el núcleo para:

- Intercambiar mensajes MQTT con AWS IoT Core.
- Intercambiar mensajes MQTT con conectores, dispositivos y otras funciones de Lambda en el grupo de Greengrass.
- Interactuar con el servicio de sombra local.
- Invocar otras funciones de Lambda locales.

- Tener acceso a recursos de secretos.
- Interactuar con el administrador de secuencias.

AWS IoT Greengrassproporciona el SDK AWS IoT Greengrass principal en los siguientes idiomas y plataformas en GitHub.

- AWS IoT Greengrass Core SDK para Java
- AWS IoT Greengrass Core SDK para Node.js
- AWS IoT Greengrass Core SDK para Python
- AWS IoT Greengrass Core SDK para C

Para incluir la dependencia del SDK de AWS IoT Greengrass Core en el paquete de implementación de funciones de Lambda:

1. Descargue el idioma o la plataforma del paquete del SDK del AWS IoT Greengrass Core que coincida con el tiempo de ejecución de su función de Lambda.
2. Descomprima el paquete descargado para obtener el SDK. El SDK es la carpeta `greengrasssdk`.
3. Incluya `greengrasssdk` en el paquete de implementación de funciones de Lambda que contiene su código de función. Este es el paquete que sube a AWS Lambda cuando crea la función de Lambda.

StreamManagerClient

Estos son los únicos SDK de AWS IoT Greengrass Core que se pueden utilizar con las operaciones del administrador de secuencias:

- SDK de Java (v 1.4.0 o posterior)
- SDK de Python (versión 1.5.0 o posterior)
- SDK de Node.js (v 1.6.0 o posterior)

Para usar el SDK de AWS IoT Greengrass Core de Python para interactuar con el administrador de secuencias, debe instalar Python 3.7 o una versión posterior. También debe instalar dependencias para incluirlas en sus paquetes de implementación de funciones de Lambda de Python:

1. Vaya al directorio de SDK que contiene el archivo de `requirements.txt`. Este archivo registra las dependencias.

2. Instale las dependencias del SDK. Por ejemplo, ejecute el siguiente comando de `pip` para instalarlas en el directorio actual:

```
pip install --target . -r requirements.txt
```

Instale el SDK de AWS IoT Greengrass Core para Python en el dispositivo del núcleo

Si está ejecutando funciones de Lambda de Python, también puede utilizar `pip` para instalar el AWS IoT Greengrass para Python en el dispositivo principal. A continuación, puede implementar las funciones sin incluir el SDK en el paquete de implementación de la función de Lambda. Para obtener más información, consulte greengrasssdk.

Este soporte está dirigido a núcleos con restricciones de tamaño. Le recomendamos que incluya el SDK en sus paquetes de implementación de funciones de Lambda cuando sea posible.

SDK de machine learning de AWS IoT Greengrass

Permite a las funciones de Lambda consumir modelos de machine learning (ML) que se implementan en el núcleo de Greengrass como recursos de ML. Las funciones de Lambda pueden usar el SDK para invocar e interactuar con un servicio de inferencia local que se implementa en el núcleo como un conector. Las funciones de Lambda y los conectores de ML también pueden usar el SDK para enviar datos al conector de ML Feedback para cargarlos y publicarlos. Para obtener más información, incluidos los ejemplos de código que utiliza el SDK, consulte the section called “Clasificación de imágenes de ML ”, the section called “Detección de objetos de ML” y the section called “ML Feedback”.

En la siguiente tabla se muestran lenguajes o plataformas admitidos para versiones de SDK y las versiones del software AWS IoT Greengrass Core en las que se pueden ejecutar.

Versión de SDK	Lenguaje o plataforma	Versión de GGC necesaria	Registro de cambios
1.1.0	Python 3.7 o 2.7	1.9.3 o posterior es	Se ha añadido compatibi

Versión de SDK	Lenguaje o plataforma	Versión de GGC necesaria	Registro de cambios
			lidad con Python 3.7 y un nuevo cliente `feedback`.
1.0.0	Python 2.7	1.7 o posteriores	Versión inicial.

Para obtener información sobre la descarga, consulte the section called “AWS IoT Greengrass Software ML SDK”.

SDK de AWS

Habilita funciones de Lambda locales para hacer llamadas directas a los servicios de AWS, como Amazon S3, DynamoDB, AWS IoT y AWS IoT Greengrass. Para utilizar un SDK de AWS en una función de Lambda de Greengrass, debe incluirlo en el paquete de implementación. Si se utiliza el SDK de AWS en el mismo paquete que el SDK de AWS IoT Greengrass Core, asegúrese de que las funciones de Lambda usan los espacios de nombres correctos. Las funciones de Lambda de Greengrass no pueden comunicarse con los servicios en la nube cuando el núcleo está desconectado.

Descargue los SDK de AWS del Centro de recursos introductorios.

Para obtener más información acerca de la creación de un paquete de implementación, consulte the section called “Crear y empaquetar una función de Lambda” en el tutorial de introducción o Creación de un paquete de implementación en la Guía del desarrollador de AWS Lambda.

Migración de funciones de Lambda basadas en la nube

El SDK de AWS IoT Greengrass Core sigue el modelo de programación del SDK de AWS, lo que facilita el traslado de las funciones de Lambda que se desarrollan para la nube a funciones de Lambda que se ejecutan en un núcleo de AWS IoT Greengrass.

Por ejemplo, la siguiente función de Lambda de Python usa el AWS SDK for Python (Boto3) a fin de publicar un mensaje en el tema `some/topic` en la nube:

```
import boto3

iot_client = boto3.client("iot-data")
response = iot_client.publish(
    topic="some/topic", qos=0, payload="Some payload".encode()
)
```

Para trasladar la función para un núcleo de AWS IoT Greengrass, en la declaración `import` y la inicialización `client`, cambie el nombre del módulo `boto3` a `greengrasssdk`, como se muestra en el siguiente ejemplo:

```
import greengrasssdk

iot_client = greengrasssdk.client("iot-data")
iot_client.publish(topic="some/topic", qos=0, payload="Some payload".encode())
```

> **Note**
>
> El SDK de AWS IoT Greengrass Core solo permite enviar mensajes MQTT con QoS = 0. Para obtener más información, consulte the section called “Mensaje de calidad del servicio”.

La similitud entre los modelos de programación también le permite desarrollar sus funciones de Lambda en la nube y luego migrarlas a AWS IoT Greengrass con un esfuerzo mínimo. Los ejecutables de Lambda no se ejecutan en la nube, por lo que no puede utilizar el SDK de AWS para desarrollarlos en la nube antes de la implementación.

Referencia a funciones de Lambda por alias o versión

Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función. Los alias se resuelven en números de versión durante la implementación del grupo. Cuando utiliza alias, la versión resuelta se actualiza a la versión a la que apunta el alias en el momento de la implementación.

AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST. Las versiones $LATEST de no están vinculadas a versiones de función publicadas, inmutables y se pueden cambiar en cualquier momento, lo cual va en contra del principio de inmutabilidad de AWS IoT Greengrass.

Una práctica habitual para mantener las funciones de Lambda Greengrass actualizadas con cambios en el código es utilizar un alias denominado **`PRODUCTION`** en su grupo y suscripciones de Greengrass. A medida que promueve nuevas versiones de su función de Lambda a producción, apunte el alias a la versión estable más reciente y, a continuación, vuelva a implementar el grupo. También puede usar este método para revertir a una versión anterior.

Control de la ejecución de funciones de Greengrass Lambda utilizando la configuración específica del grupo

AWS IoT Greengrass permite la administración basada en la nube de funciones de . Aunque el código y las dependencias de una función de Lambda se administran mediante AWS Lambda, puede configurar el comportamiento de la función de Lambda cuando se ejecuta en un grupo de Greengrass.

Configuración específica del grupo

AWS IoT Greengrass proporciona la siguiente configuración específica del grupo para las funciones de Greengrass Lambda.

Usuario del sistema y grupo

> La identidad de acceso que se utiliza para ejecutar una función de Lambda. De forma predeterminada, las funciones de Lambda se ejecutan como la identidad de acceso predeterminada del grupo. Por lo general, son las cuentas del sistema AWS IoT Greengrass estándar (ggc_user y ggc_group). Puede cambiar la configuración y elegir el ID de usuario y el ID de grupo con los permisos necesarios para ejecutar la función de Lambda. Puede anular tanto UID como GID o uno solo si deja el otro campo en blanco. Esta configuración le ofrece un control más detallado sobre el acceso a recursos de dispositivos. Le recomendamos que configure su hardware de Greengrass con los límites de recursos adecuados, permisos de archivos y cuotas de disco para los usuarios y grupos cuyos permisos se utilizan para la ejecución de funciones de Lambda.
>
> Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

> **Important**
>
> Se recomienda evitar la ejecución de funciones de Lambda como raíz a menos que sea absolutamente necesario. Si se ejecuta como raíz, se incrementan los siguientes riesgos:
>
> - El riesgo de cambios no intencionados, como la eliminación accidental de un archivo crítico.
> - El riesgo que representan personas malintencionadas para sus datos y dispositivos.
> - El riesgo de que un contenedor se escape cuando los contenedores de Docker funcionan con `--net=host` y `UID=EUID=0`.
>
> Si tiene que ejecutar como usuario raíz, debe actualizar la configuración de AWS IoT Greengrass para habilitarlo. Para obtener más información, consulte the section called "Ejecución de una función de Lambda como raíz".

ID de usuario del sistema (número)

El ID de usuario para el usuario que tiene los permisos necesarios para ejecutar la función de Lambda. Esta opción solo está disponible si elige Ejecutar como otro ID de usuario/ID de grupo. Puede utilizar el comando de getent passwd en el núcleo de su dispositivo de AWS IoT Greengrass para buscar el ID del usuario que desea usar para ejecutar la función de Lambda.

Si usa el mismo ID para ejecutar los procesos y la función de Lambda en un dispositivo de Greengrass core, su rol de grupo de Greengrass puede otorgar credenciales temporales para los procesos. Los procesos pueden usar las credenciales temporales en todas las implementaciones de Greengrass core.

ID de grupo del sistema (número)

El ID de grupo para el grupo que tiene los permisos necesarios para ejecutar la función de Lambda. Esta opción solo está disponible si elige Ejecutar como otro ID de usuario/ID de grupo. Puede utilizar el comando de getent group en el núcleo de su dispositivo de AWS IoT Greengrass para buscar el ID del usuario que desea usar para ejecutar la función de Lambda.

Contenedorización de funciones de Lambda

Elija si la función de Lambda se ejecuta con la creación de contenedores predeterminada para el grupo o especifique la creación de contenedores que se debe utilizar siempre para esta función de Lambda.

El modo de creación de contenedores de una función de Lambda determina su nivel de aislamiento.

- Las funciones de Lambda en contenedor se ejecutan en el modo Contenedor de Greengrass. La función de Lambda se ejecuta en un entorno de tiempo de ejecución (o espacio de nombres) aislado dentro del contenedor de AWS IoT Greengrass.
- Las funciones de Lambda que no están en un contenedor se ejecutan en el modo Sin contenedor. Las funciones de Lambda se ejecutan como un proceso normal de Linux sin ningún tipo de aislamiento.

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

Le recomendamos que ejecute las funciones de Lambda en un contenedor de Greengrass a menos que su caso de uso requiera la ejecución sin creación de contenedores. Cuando las funciones de Lambda se ejecutan en un contenedor de Greengrass, puede utilizar recursos de dispositivos y locales asociados, así como disfrutar de los beneficios del aislamiento y de una mayor seguridad. Antes de cambiar la creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.

> **Note**
>
> Para ejecutar sin habilitar su espacio de nombres de kernel de dispositivo y cgroup, todas las funciones de Lambda se deben ejecutar sin creación de contenedores. Puede conseguirlo fácilmente configurando la creación de contenedores predeterminada para el grupo. Para obtener más información, consulte the section called “Configuración de la creación de contenedores predeterminada para funciones Lambda de un grupo”.

Memory limit (Límite de memoria)

Asignación de memoria para la función. El valor predeterminado es 16 MB.

> **Note**
>
> La configuración del límite de memoria se descarta al cambiar la función de Lambda para ejecución sin creación de contenedores. Las funciones de Lambda que se ejecutan sin contenedorización no tienen límite de memoria. La configuración del límite de memoria se descarta al cambiar la función Lambda o la configuración de creación de contenedores predeterminada del grupo para ejecutarse sin creación de contenedores.

Timeout (Tiempo de espera)

La cantidad de tiempo antes de que se termine la función o solicitud. El valor predeterminado es de 3 segundos.

Anclado

El ciclo de vida de una función de Lambda puede ser bajo demanda o de larga duración. El valor predeterminado es bajo demanda.

Una función de Lambda bajo demanda se inicia en un contenedor nuevo o reutilizado cuando se invoca. Las solicitudes a la función puede ser procesados por cualquier contenedor disponible. Una función de Lambda de larga duración se inicia automáticamente después de iniciarse AWS IoT Greengrass y se mantiene en ejecución en su propio contenedor (o entorno de pruebas). Todas las solicitudes a la función se procesan en el mismo contenedor. Para obtener más información, consulte the section called “Configuración del ciclo de vida”.

Read access to /sys directory (Acceso de lectura al directorio /sys)

Si la función puede obtener acceso o no a la carpeta /sys del host. Utilícelo cuando la función deba leer información del dispositivo desde /sys. El valor predeterminado es false.

> **Note**
>
> Esta configuración no está disponible cuando se ejecuta de una función de Lambda sin creación de contenedores. El valor de esta configuración se descarta al cambiar la función de Lambda para que se ejecute sin contenedores.

Tipo de codificación

El tipo de codificación esperado de la carga de entrada de la función, JSON o binaria. El valor predeterminado es JSON.

Se admite el tipo de codificación binaria a partir del software de AWS IoT Greengrass Core v1.5.0 y el SDK de AWS IoT Greengrass Core v1.1.0. Aceptar datos de entrada binarios puede ser útil para funciones que interactúan con datos de los dispositivos, ya que las limitadas capacidades de hardware de estos a menudo hacen que sea difícil o imposible construir un tipo de datos JSON.

> **Note**
>
> Los Ejecutables de Lambda solo admiten el tipo de codificación binaria (no JSON).

Argumentos del proceso

Los argumentos de línea de comandos para pasar a la función de Lambda cuando se ejecuta.

Variables de entorno

Pares clave-valor que pueden pasar ajustes de forma dinámica al código y bibliotecas de la función. Las variables de entorno local funcionan de la misma forma que las variables de entorno de las funciones de AWS Lambda, pero están disponibles en el entorno de núcleo.

Políticas de acceso a recursos

Una lista de hasta 10 recursos locales, recursos secretos y recursos de aprendizaje automático a los que puede acceder la función de Lambda y los correspondientes permisos `read-only` o `read-write`. En la consola, estos recursos afiliados aparecen en la página de configuración del grupo, en la pestaña Recursos.

El modo de creación de contenedores afecta a la forma en que las funciones de Lambda pueden acceder a los recursos de volumen y dispositivos locales y a los recursos de aprendizaje automático.

- Las funciones de Lambda que no están en un contenedor deben acceder directamente a los recursos de volumen y dispositivos locales a través del sistema de archivos del dispositivo central.
- Para permitir que las funciones de Lambda que no están en un contenedor tengan acceso a los recursos de aprendizaje automático del grupo de Greengrass, debe establecer las propiedades de permisos de acceso y de propietario del recurso en el recurso de aprendizaje automático. Para obtener más información, consulte the section called “Acceso a recursos de aprendizaje automático”.

Para obtener información sobre el uso de la AWS IoT Greengrass API para establecer valores de configuración específicos de grupo para las funciones Lambda definidas por el usuario, CreateFunctionDefinitionconsulte la Referencia de AWS IoT Greengrass Version 1la API create-function-definitiono la Referencia de comandos. AWS CLI Para implementar funciones de Lambda en un núcleo de Greengrass, cree una versión de definición de función que contenga sus funciones,

cree una versión de grupo que haga referencia a la versión de definición de función y a otros componentes del grupo y, a continuación, implemente el grupo.

Ejecución de una función de Lambda como raíz

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

Antes de poder ejecutar una o varias funciones de Lambda como raíz, primero debe actualizar la configuración de AWS IoT Greengrass para habilitar el soporte. El soporte para ejecutar funciones de Lambda como raíz está desactivado de forma predeterminada. La implementación genera un error si intenta implementar una función de Lambda y ejecutarla como raíz (UID y GID de 0) y no ha actualizado la configuración de AWS IoT Greengrass . Un error como el siguiente aparece en el registro de tiempo de ejecución (*greengrass_root*/ggc/var/log/system/runtime.log):

```
lambda(s)
[list of function arns] are configured to run as root while Greengrass is not
 configured to run lambdas with root permissions
```

Important

Se recomienda evitar la ejecución de funciones Lambda como raíz a menos que sea absolutamente necesario. Si se ejecuta como raíz, se incrementan los siguientes riesgos:

- El riesgo de cambios no intencionados, como la eliminación accidental de un archivo crítico.
- El riesgo que representan personas malintencionadas para sus datos y dispositivos.
- El riesgo de que un contenedor se escape cuando los contenedores de Docker funcionan con `--net=host` y `UID=EUID=0`.

Para permitir que las funciones de Lambda se ejecuten como raíz

1. En el dispositivo de AWS IoT Greengrass, acceda a la carpeta *greengrass-root*/config.

   Note

   De forma predeterminada, *greengrass-root* es el directorio /greengrass.

2. Modifique el archivo config.json para añadir `"allowFunctionsToRunAsRoot" : "yes"` al campo `runtime`. Por ejemplo:

```
{
  "coreThing" : {
    ...
  },
  "runtime" : {
    ...
    "allowFunctionsToRunAsRoot" : "yes"
  },
  ...
}
```

3. Use los siguientes comandos para reiniciar AWS IoT Greengrass:

```
cd /greengrass/ggc/core
sudo ./greengrassd restart
```

   Ahora puede establecer el ID de usuario y el ID de grupo (UID/GID) de las funciones de Lambda en 0 para ejecutar la función de Lambda como raíz.

Puede cambiar el valor de `"allowFunctionsToRunAsRoot"` a `"no"` y reiniciar AWS IoT Greengrass si desea desactivar que las funciones de Lambda se ejecuten como raíz.

Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

De forma predeterminada, las funciones de Lambda se ejecutan dentro de un contenedor AWS IoT Greengrass. Dicho contenedor proporciona aislamiento entre las funciones y el host, lo que brinda mayor seguridad tanto al host como a las funciones del contenedor.

Le recomendamos que ejecute las funciones de Lambda en un contenedor de Greengrass a menos que su caso de uso requiera la ejecución sin creación de contenedores. Al ejecutar las funciones de Lambda en un contenedor Greengrass, tiene más control sobre la restricción de acceso a los recursos.

A continuación, se muestran algunos casos de uso de ejemplo para la ejecución sin creación de contenedores:

- Desea ejecutar AWS IoT Greengrass en un dispositivo que no admite el modo de contenedor (por ejemplo, porque utiliza una distribución especial de Linux o porque tiene una versión del kernel demasiado antigua).
- Desea ejecutar su función de Lambda en otro entorno de contenedor con su propio OverlayFS, pero detecta conflictos de OverlayFS cuando la ejecuta en un contenedor de Greengrass.
- Necesita acceso a recursos locales con rutas que no se pueden determinar en el momento de la implementación o cuyas rutas pueden cambiar tras la implementación, como en el caso de algunos dispositivos conectables.
- Tiene una aplicación heredada que se ha escrito como y ha detectado problemas cuando se ejecuta como una función de Lambda en un contenedor.

Diferencias en la creación de contenedores

Creación de contenedores	Notas
Contenedor de Greengrass	• Todas las características de AWS IoT Greengrass están disponibles cuando ejecuta una función de Lambda en un contenedor de Greengrass. • Las funciones de Lambda que se ejecutan en un contenedor Greengrass no tienen acceso al código implementado de otras funciones de Lambda, incluso aunque se ejecuten con el mismo ID de grupo. Es decir, las funciones de Lambda se ejecutan con mayor aislamien to de entre sí. • Dado que las funciones de Lambda que se ejecutan en un contenedor AWS IoT Greengrass ejecutan todos los procesos secundarios en el mismo contenedor que la función de Lambda, los procesos secundari os se terminan cuando se termina la función de Lambda.

Creación de contenedores	Notas

Sin contenedor	• Las siguientes características no están disponibles para funciones de Lambda no incluidas en contenedores:
	  • Límites de memoria de funciones de Lambda.
	  • Dispositivos locales y recursos de volumen. Debe obtener acceso a estos recursos directamente en el dispositivo principal en lugar de obtener acceso a ellos como miembros del grupo de Greengrass.
	• Si la función de Lambda que no está en un contenedor accede a un recurso de aprendizaje automático, debe identificar a un propietario del recurso y establecer permisos de acceso en el recurso, no en la función de Lambda. Este requiere AWS IoT Greengrass Core versión 1.10 o posterior. Para obtener más información, consulte the section called “Acceso a recursos de aprendizaje automático”.
	• La función de Lambda tiene acceso de solo lectura al código implementado de otras funciones de Lambda que se están ejecutando con el ID del mismo grupo.
	• No terminan automáticamente las funciones de Lambda que abarcan procesos secundarios en una sesión de proceso distinta o con un controlador SIGHUP (signal hangup) anulado porAWS IoT Greengrass, como la utilidad nohup, cuando se termina la función de Lambda principal.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo Greengrass no se aplica a los conectores.

El cambio de la creación de contenedores para una función de Lambda puede provocar problemas durante la implementación. Si había asignado recursos locales a la función de Lambda que ya no están disponibles con la nueva configuración de creación de contenedores, la implementación genera un error.

- Cuando se cambia una función de Lambda de ejecución en un contenedor de Greengrass a ejecución sin creación de contenedores, se descartan los límites de memoria de la función. Debe acceder al sistema de archivos directamente en lugar de utilizar los recursos locales asociados. Debe eliminar los recursos asociados antes de la implementación.
- Al cambiar una función de Lambda de la ejecución sin creación de contenedores a la ejecución en un contenedor, la función de Lambda pierde el acceso directo al sistema de archivos. Debe definir un límite de memoria para cada función de o aceptar la opción predeterminada de 16 MB. Puede establecer esta configuración para cada función de Lambda antes de la implementación.

Para cambiar la configuración de creación de contenedores para una función de Lambda

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija el grupo que contiene la función de Lambda cuya configuración desea cambiar.
3. Elija la tabla de las funciones Lambda.
4. En la función de Lambda que desee cambiar, elija los puntos suspensivos (…) y, a continuación, elija Editar configuración.
5. Cambiar la configuración de creación de contenedores. Si configura la función de Lambda para que ejecute un contenedor de Greengrass, también deberá configurar las propiedades Límite de memoria y Acceso de lectura al directorio /sys.
6. Seleccione Guardar y, a continuación, Confirmar para guardar los cambios en la función de Lambda.

Los cambios surten efecto cuando el grupo se implementa.

También puede utilizar la referencia CreateFunctionDefinitiony CreateFunctionDefinitionVersionen la API. AWS IoT Greengrass Si va a cambiar la configuración de creación de contenedores, asegúrese de actualizar los demás parámetros también. Por ejemplo, si va a cambiar entre la ejecución de una función de Lambda en un contenedor de Greengrass a la ejecución sin creación de contenedores, asegúrese de borrar el parámetro `MemorySize`.

Determinación de los modos de aislamiento admitidos en el dispositivo de Greengrass

Puede utilizar el comprobador de dependencias de AWS IoT Greengrass para determinar los modos de aislamiento (contenedor de Greengrass/sin contenedor) admitidos por su dispositivo de Greengrass.

Para ejecutar el comprobador de dependencias de AWS IoT Greengrass

1. Descarga y ejecuta el comprobador de AWS IoT Greengrass dependencias desde el GitHubrepositorio.

   ```
   wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
   unzip greengrass-dependency-checker-GGCv1.11.x.zip
   cd greengrass-dependency-checker-GGCv1.11.x
   sudo modprobe configs
   sudo ./check_ggc_dependencies | more
   ```

2. Si aparece `more`, pulse la tecla Spacebar para mostrar otra página de texto.

Para obtener más información acerca del comando modprobe, ejecute man modprobe en el terminal.

Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo

Esta característica está disponible para AWS IoT Greengrass Core versión 1.8 y posteriores.

Para tener un mayor control sobre el acceso a los recursos del dispositivo, puede configurar la identidad de acceso predeterminada que se utiliza para ejecutar funciones de Lambda en el grupo. Esta configuración determina los permisos predeterminados que se han asignado a sus funciones de Lambda cuando se ejecutan en el dispositivo central. Para invalidar la configuración de las distintas funciones del grupo, puede utilizar la propiedad Run as (Ejecutar como) de la función. Para obtener más información, consulte Ejecutar como.

Esta configuración de nivel de grupo se utiliza también para ejecutar el software de AWS IoT Greengrass Core subyacente. Se compone de funciones de Lambda del sistema que administran operaciones, como el redireccionamiento de mensajes, la sincronización de instantáneas locales y la detección automática de direcciones IP.

La identidad de acceso predeterminada se puede configurar para que se ejecute como las cuentas de AWS IoT Greengrass estándar (ggc_user y ggc_group) o para que utilice los permisos de otro usuario o grupo. Le recomendamos que configure su hardware de Greengrass con los límites de recursos, permisos de archivos y cuotas de disco adecuados para los usuarios y grupos cuyos permisos se utilizan para la ejecución de funciones de Lambda del sistema o definidas por el usuario.

Para modificar la identidad de acceso predeterminada para su grupo de AWS IoT Greengrass

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija el grupo cuya configuración desea cambiar.
3. Seleccione la pestaña Funciones de Lambda y, en la sección Entorno de tiempo de ejecución de las funciones de Lambda predeterminado, seleccione Editar.
4. En la página de Editar entorno de tiempo de ejecución Lambda predeterminado para Usuario y grupo del sistema predeterminados, elija Otro ID de usuario o grupo.

   Al elegir esta opción, se muestran los campos System user D (number) y System group ID (number).
5. Escriba un ID de usuario, un ID de grupo o ambos. Si deja un campo en blanco, se utiliza la cuenta del sistema de Greengrass correspondiente (ggc_user o ggc_group).

   - En ID del usuario del sistema (número), especifique el ID del usuario que tiene los permisos que desea utilizar de forma predeterminada para ejecutar funciones de Lambda en el grupo. Puede utilizar el comando getent passwd en el dispositivo de AWS IoT Greengrass para buscar el ID de usuario.
   - En ID del grupo del sistema (número), especifique el ID del grupo que tiene los permisos que desea utilizar de forma predeterminada para ejecutar funciones de Lambda en el grupo. Puede utilizar el comando getent group en el dispositivo de AWS IoT Greengrass para buscar el ID de grupo.

> **Important**
>
> La ejecución como usuario raíz aumenta los riesgos en los datos y el dispositivo. No realice ejecuciones como usuario raíz (UID/GID=0) a menos que su caso de negocio lo requiera. Para obtener más información, consulte the section called “Ejecución de una función de Lambda como raíz”.

Los cambios surten efecto cuando el grupo se implementa.

Configuración de la creación de contenedores predeterminada para funciones Lambda de un grupo

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

La configuración de creación de contenedores de un grupo de Greengrass determina la creación de contenedores predeterminada para las funciones de Lambda del grupo.

- En el modo Contenedor de Greengrass, las funciones de Lambda se ejecutan en un entorno de tiempo de ejecución aislado dentro de los contenedores de AWS IoT Greengrass de forma predeterminada.
- En el modo Sin contenedor, las funciones de Lambda se ejecutan como procesos normales de Linux de forma predeterminada.

Puede modificar la configuración del grupo para especificar la creación de contenedores predeterminada para las funciones de Lambda del grupo. Puede pasar por alto esta configuración para una o varias funciones de Lambda en el grupo si desea que las funciones de Lambda se ejecuten con una creación de contenedores distinta a la del grupo predeterminado. Antes de cambiar la configuración de la creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.

> **Important**
>
> Si desea cambiar la creación de contenedores predeterminada del grupo, pero tiene una o varias funciones que utilizan una creación de contenedores distinta, cambie la configuración de las funciones de Lambda antes de cambiar la configuración del grupo. Si cambia primero

la configuración de creación de contenedores del grupo, se descartan los valores de la configuración Memory limit (Límite de memoria) y Read access to /sys directory (Acceso de lectura al directorio /sys).

Para modificar la configuración de creación de contenedores de su grupo de AWS IoT Greengrass

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija el grupo cuya configuración desea cambiar.
3. Elija la pestaña Función de Lambda.
4. En Tiempo de ejecución predeterminado de funciones Lambda, elija Editar.
5. En la página Tiempo de ejecución predeterminado de funciones de Lambda, en Contenedorización de la función de Lambda predeterminada, cambie el ajuste de contenedorización.
6. Seleccione Guardar.

Los cambios surten efecto cuando el grupo se implementa.

Flujos de comunicación para las funciones de Lambda de Greengrass

Las funciones de Lambda de Greengrass pueden utilizar diversos métodos para comunicarse con otros miembros del grupo de AWS IoT Greengrass, servicios locales y servicios en la nube (incluidos los servicios de AWS).

Comunicación con mensajes MQTT

Las funciones de Lambda pueden enviar y recibir mensajes MQTT utilizando un patrón publish-subscribe (publicar-suscribir) que se controla a través de las suscripciones.

Este flujo de comunicación permite que las funciones de Lambda intercambien mensajes con las siguientes entidades:

- Dispositivos cliente en el grupo.

- Conectores en el grupo.
- Otras funciones de Lambda del grupo.
- AWS IoT.
- Servicio de sombra del dispositivo local.

Las suscripciones especifican un origen del mensaje, un destino del mensaje y un tema (o asunto), que se utilizan para direccionar los mensajes desde el origen hasta el destino. Los mensajes que se publican en una función de Lambda se transfieren al controlador registrado de la función. Las suscripciones permiten más seguridad y proporcionan interacciones predecibles. Para obtener más información, consulte the section called “Suscripciones administradas en el flujo de trabajo de mensajería de MQTT”.

> **Note**
>
> Cuando el dispositivo central está sin conexión, las funciones de Lambda de Greengrass pueden intercambiar mensajes con otros dispositivos cliente, conectores, otras funciones y sombras locales, pero los mensajes dirigidos a AWS IoT se ponen en cola. Para obtener más información, consulte the section called “Cola de mensajes MQTT”.

Otros flujos de comunicación

- Para interactuar con los modelos de machine learning y los recursos de volumen y dispositivos locales en un dispositivo central, las funciones de Lambda de G utilizan las interfaces del sistema operativo específico de la plataforma. Por ejemplo, puede utilizar el método de `open` en el módulo os en las funciones de Python 2.7. Para permitir que una función acceda a un recurso, la función debe estar asociada con el recurso y tener el permiso `read-only` o `read-write`. Para obtener más información, incluida la disponibilidad de las versiones principales de AWS IoT Greengrass, consulte Acceder a recursos locales y the section called “Acceso a recursos de machine learning desde el código de la función de Lambda”.

  > **Note**
  >
  > Si ejecuta la función de Lambda sin creación de contenedores, no puede utilizar los recursos de volumen y dispositivos locales asociados, y debe acceder a dichos recursos directamente.

- Las funciones de Lambda pueden utilizar el cliente de `Lambda` en el SDK de AWS IoT Greengrass Core para invocar otras funciones de Lambda en el grupo de Greengrass.
- Las funciones de Lambda pueden usar el SDK de AWS para comunicarse con los servicios de AWS. Para obtener más información, consulte SDK de AWS.
- Las funciones de Lambda pueden usar interfaces de terceros para comunicarse con servicios en la nube externos, de modo similar a las funciones de Lambda basadas en la nube.

> **Note**
>
> Las funciones de Lambda de Greengrass no pueden comunicarse con AWS u otros servicios en la nube cuando el núcleo está desconectado.

Recuperación del tema (o asunto) de MQTT de entrada

AWS IoT Greengrass utiliza suscripciones para controlar el intercambio de mensajes MQTT entre dispositivos de cliente, funciones de Lambda y conectores en un grupo, y con AWS IoT o el servicio de sombras local. Las suscripciones definen el origen del mensaje, el destino del mensaje y el tema de MQTT utilizados para dirigir los mensajes. Cuando el destino es una función de Lambda, el controlador de la función se invoca cuando el origen publica un mensaje. Para obtener más información, consulte the section called “Comunicación con mensajes MQTT”.

En el siguiente ejemplo, se muestra cómo una función de Lambda puede obtener el tema de entrada desde el `context` pasado al controlador. Para ello, obtiene acceso a la clave `subject` desde la jerarquía del contexto (`context.client_context.custom['subject']`). El ejemplo analiza también el mensaje JSON de entrada y, a continuación, publica el tema analizado y el mensaje.

> **Note**
>
> En la API de AWS IoT Greengrass, el tema de una suscripción se representa mediante la propiedad `subject`.

```
import greengrasssdk
import logging

client = greengrasssdk.client('iot-data')
```

```
OUTPUT_TOPIC = 'test/topic_results'

def get_input_topic(context):
    try:
        topic = context.client_context.custom['subject']
    except Exception as e:
        logging.error('Topic could not be parsed. ' + repr(e))
    return topic

def get_input_message(event):
    try:
        message = event['test-key']
    except Exception as e:
        logging.error('Message could not be parsed. ' + repr(e))
    return message

def function_handler(event, context):
    try:
        input_topic = get_input_topic(context)
        input_message = get_input_message(event)
        response = 'Invoked on topic "%s" with message "%s"' % (input_topic,
 input_message)
        logging.info(response)
    except Exception as e:
        logging.error(e)

    client.publish(topic=OUTPUT_TOPIC, payload=response)

    return
```

Para probar la función, añádala a su grupo utilizando las opciones de configuración predeterminadas. A continuación, añada las siguientes suscripciones e implemente el grupo. Para obtener instrucciones, consulte the section called “Módulo 3 (primera parte): Funciones de Lambda en AWS IoT Greengrass”.

Fuente	
de	
temas	

Filtro de temas (input_message de IoT)	test/ topic (Nube de IoT)
test/ topic_ results (Nube de IoT)	

Una vez completada la implementación, invoque la función.

1. En la consola de AWS IoT, abra la página del cliente de pruebas MQTT.
2. Para suscribirse al tema `test/topic_results`, seleccione la pestaña Suscribirse a un tema.
3. Publique un mensaje en el tema `test/input_message` seleccionando la pestaña Publicar en un tema. En este ejemplo, debe incluir la propiedad `test-key` en el mensaje JSON.

   ```
   {
     "test-key": "Some string value"
   }
   ```

   Si todo es correcto, la función publica el tema de entrada y la cadena del mensaje en el tema `test/topic_results`.

Configuración del ciclo de vida de las funciones de Lambda de Greengrass

El ciclo de vida de la función de Lambda de Greengrass determina cuándo se inicia una función y cómo crea y utiliza contenedores. El ciclo de vida también determina cómo se conservan las variables y la lógica de procesamiento previo que está fuera del controlador de la función.

AWS IoT Greengrass admite ciclos de vida bajo demanda (predeterminado) o de larga duración.

- Las funciones bajo demanda se inician cuando se invocan y se detienen cuando no quedan tareas que ejecutar. Una invocación de la función crea un contenedor independiente (o entorno de pruebas) para procesar invocaciones, a menos que haya un contenedor disponible que se pueda reutilizar. Los datos que se envíe a la función pueden extraerse de cualquiera de los contenedores.

  Es posible ejecutar en paralelo varias invocaciones de una función bajo demanda.

  No se conserva ninguna variable ni lógica de procesamiento previo que se defina fuera del controlador de la función cuando se crean nuevos contenedores.

- Las funciones de larga duración (o adjuntas) se inician automáticamente cuando se inicia el núcleo de AWS IoT Greengrass y se ejecutan en un solo contenedor. Todos los datos que se envían a la función los extrae el mismo contenedor.

  Hay varias invocaciones en la cola hasta que se ejecuten las invocaciones anteriores.

  Las variables y lógica de procesamiento previo que se definen fuera del controlador de la función se conservan para cada invocación del controlador.

  Las funciones de Lambda de larga duración resultan útiles cuando necesita empezar a trabajar sin ninguna entrada inicial. Por ejemplo, una función de larga duración puede cargar e iniciar el procesamiento de un modelo de ML para que esté listo para cuando la función empiece a recibir los datos del dispositivo.

  > **Note**
  >
  > Recuerde que las funciones de larga duración tienen tiempos de espera que están asociados con invocaciones de su controlador. Si desea ejecutar de forma indefinida el código en ejecución, debe iniciarlo fuera del controlador. Asegúrese de que no haya código de bloqueo fuera del controlador que pudiera impedir la inicialización de la función. Estas funciones se ejecutan a menos que el núcleo se detenga (por ejemplo, al reiniciarse una implementación de grupo o un dispositivo) o que la función adopte un estado de error (por ejemplo, un error de tiempo de espera del controlador, una excepción no detectada o si se superan los límites de memoria).

Para obtener más información acerca de la reutilización de contenedores, consulte Comprender la reutilización de contenedores AWS Lambda en el AWSBlog de informática de .

Ejecutables de Lambda

Esta característica está disponible para AWS IoT Greengrass Core versión 1.6 y posteriores.

Un ejecutable es un tipo de función de Lambda de Greengrass que puede utilizar para ejecutar código binario en el entorno del núcleo. Le permite ejecutar la funcionalidad específica del dispositivo de manera nativa y se beneficia de una menor huella de código compilado. Los ejecutables de Lambda pueden ser invocados por eventos, pueden invocar otras funciones y pueden acceder a recursos locales.

Los ejecutables de Lambda admite solo el tipo de cifrado binario (no JSON) pero, por lo demás, puede administrarlos en su grupo de Greengrass e implementarlos como otras funciones de Lambda de Greengrass. Sin embargo, el proceso de creación de Lambda es diferente de la creación de funciones de Lambda, Python, Java y Node.js:

- No puede utilizar la consola de AWS Lambda para crear (o gestionar) un ejecutable de Lambda. Solo puede crear un ejecutable de Lambda mediante la API de AWS Lambda.
- Puede cargar el código de la función a AWS Lambda como un ejecutable compilado que incluye el SDK de AWS IoT Greengrass Core para C..
- Especificará el nombre del ejecutable como el controlador de la función.

Los ejecutables de Lambda deben implementar determinadas llamadas y patrones de programación en su código de la función. Por ejemplo, el método `main` debe:

- Llamar a `gg_global_init` para inicializar variables globales internas de Greengrass. Esta función debe llamarse antes de crear subprocesos y antes de llamar a cualquier otra función de SDK de AWS IoT Greengrass Core.
- Llame a `gg_runtime_start` para registrar el controlador de la función con el tiempo de ejecución de Lambda de Greengrass. Esta función debe llamarse durante la inicialización. Al llamar a esta función el tiempo de ejecución utiliza el subproceso actual. El parámetro `GG_RT_OPT_ASYNC` opcional indica a esta función que no bloquee, sino que cree un nuevo subproceso para el tiempo de ejecución. Esta función utiliza un controlador `SIGTERM`.

El siguiente fragmento es el `main` método utilizado en el ejemplo de código simple_handler.c en adelante. GitHub

```
int main() {
```

```
    gg_error err = GGE_SUCCESS;

    err = gg_global_init(0);
    if(err) {
        gg_log(GG_LOG_ERROR, "gg_global_init failed %d", err);
        goto cleanup;
    }

    gg_runtime_start(handler, 0);

cleanup:
    return -1;
}
```

Para obtener más información acerca de los requisitos exigidos, las restricciones y otros detalles de implementación, consulte SDK de AWS IoT Greengrass Core para C.

Crear un ejecutable de Lambda

Después de compilar el código junto con el SDK, utilice la API de AWS Lambda para crear una función de Lambda y cargar un ejecutable compilado.

> **Note**
>
> La función deberá elaborarse con un compilador C89 compatible.

En el siguiente ejemplo se utiliza el comando de CLI create-function para crear un ejecutable de Lambda. El comando especifica:

- El nombre del ejecutable para el controlador. Debe ser el nombre exacto del ejecutable compilado.
- La ruta del archivo `.zip` que contiene el ejecutable compilado.
- `arn:aws:greengrass:::runtime/function/executable` para el tiempo de ejecución. Este es el tiempo de ejecución para todos los ejecutables de Lambda.

> **Note**
>
> Para `role` puede especificar el ARN de cualquier rol de ejecución de Lambda. AWS IoT Greengrass no utiliza este rol, pero el parámetro es necesario para crear la función. Para

obtener más información sobre los roles de ejecución de Lambda, consulte modelo de permisos de AWS Lambda en la Guía para desarrolladores de AWS Lambda.

```
aws lambda create-function \
--region aws-region \
--function-name function-name \
--handler executable-name \
--role role-arn \
--zip-file fileb://file-name.zip \
--runtime arn:aws:greengrass:::runtime/function/executable
```

A continuación, utilice la API de AWS Lambda para publicar una versión y crear un alias.

- Utilice publish-version para publicar una versión de la función.

  ```
  aws lambda publish-version \
  --function-name function-name \
  --region aws-region
  ```

- Utilice create-alias para crear un alias que apunte a la versión que acaba de publicar. Recomendamos que el alias haga referencia a funciones de Lambda cuando las añade a un grupo de Greengrass.

  ```
  aws lambda create-alias \
  --function-name function-name \
  --name alias-name \
  --function-version version-number \
  --region aws-region
  ```

> **Note**
>
> La consola de AWS Lambda no muestra los ejecutables de Lambda. Para actualizar el código de la función, debe utilizar la API de AWS Lambda.

A continuación, añada el ejecutable de Lambda a un grupo de Greengrass, configúrelo para aceptar los datos de entrada binarios en su configuración específica del grupo e implemente el grupo. Puede hacerlo en la consola de AWS IoT Greengrass o utilizando la API AWS IoT Greengrass.

Ejecución de AWS IoT Greengrass en un contenedor Docker

AWS IoT Greengrass se puede configurar para que se ejecute en un contenedor de Docker.

Puede descargar un Dockerfile a través de Amazon CloudFront que tenga el software y las dependencias de AWS IoT Greengrass Core instalados. Para modificar la imagen de Docker de manera que se ejecute en arquitecturas de plataforma distintas o para reducir el tamaño de la imagen de Docker, consulte el archivo `README` del paquete de descarga de Docker.

Para ayudarle a comenzar a experimentar con AWS IoT Greengrass, AWS también proporciona imágenes de Docker prediseñadas que tienen instalado el software AWS IoT Greengrass Core y las dependencias. Puede descargar imágenes de Docker Hub o Amazon Elastic Container Registry (Amazon ECR). Estas imágenes prediseñadas utilizan imágenes base de Amazon Linux 2 (x86_64) y Alpine Linux (x86_64, Armv7l o AArch64).

> **Important**
>
> El 30 de junio de 2022, AWS IoT Greengrass finalizó el mantenimiento de las imágenes de Docker del software AWS IoT Greengrass Core v1.x publicadas en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub. Puede seguir descargando estas imágenes de Docker desde Amazon ECR y Docker Hub hasta el 30 de junio de 2023, es decir, un año después de que finalice el mantenimiento. Sin embargo, las imágenes de Docker de la versión 1.x del software AWS IoT Greengrass Core ya no reciben parches de seguridad ni correcciones de errores una vez finalizado el mantenimiento el 30 de junio de 2022. Si ejecuta una carga de trabajo de producción que depende de estas imágenes de Docker, le recomendamos que cree sus propias imágenes de Docker con los archivos Docker que AWS IoT Greengrass proporciona. Para obtener más información, consulte AWS IoT Greengrass Software Docker.

En este tema se describe cómo descargar la imagen de DockerAWS IoT Greengrass desde Amazon ECR y ejecutarla en plataformas Windows, macOS y Linux (x86_64). El tema contiene los siguientes pasos:

1. Obtener la imagen del contenedor AWS IoT Greengrass de Amazon ECR
2. Crear y configurar el grupo y el núcleo de Greengrass
3. Ejecutar AWS IoT Greengrass localmente

4. Configurar la creación de contenedores "Sin contenedor" para el grupo
5. Implementar funciones de Lambda en el contenedor de Docker
6. (Opcional) Implementar los dispositivos cliente que interactúan con Greengrass en el contenedor de Docker

Las siguientes características no se admiten cuando se ejecuta AWS IoT Greengrass en un contenedor Docker:

- Conectores que se ejecutan en el modo Contenedor de Greengrass. Para ejecutar un conector en un contenedor de Docker, el conector debe ejecutarse en modo Sin contenedor. Para buscar conectores compatibles con el modo Sin contenedor, consulte the section called “conectores de Greengrass proporcionados por AWS”. Algunos de estos conectores tienen un parámetro de modo de aislamiento que debe establecer en Sin contenedor.
- Dispositivos locales y recursos de volumen. Las funciones de Lambda definidas por el usuario que se ejecutan en el contenedor de Docker deben obtener acceso directamente a los dispositivos y volúmenes del dispositivo principal.

Estas características no son compatibles cuando el entorno de tiempo de ejecución de Lambda para el grupo Greengrass se establece en Sin contenedor, que es necesario para ejecutar AWS IoT Greengrass en un contenedor de Docker.

Requisitos previos

Antes de empezar este tutorial, debe hacer lo siguiente.

- Debe instalar el software y las versiones siguientes en su ordenador host en función de la versión AWS Command Line Interface (AWS CLI) que elija.

  AWS CLI version 2

  - Docker, versión 18.09 o superior. Es posible que las versiones anteriores también funcionen, pero recomendamos la 18.09 o una versión posterior.
  - AWS CLI versión 2.0.0 o posterior
    - Para instalar la versión 2 de AWS CLI, consulte Instalación de la AWS CLI versión 2.
    - Para configurar el AWS CLI, consulte Configuración de la AWS CLI.

> **Note**
>
> Para actualizar a una versión 2 de AWS CLI posterior en un equipo con Windows, debe repetir el proceso de instalación de MSI.

AWS CLI version 1

- Docker, versión 18.09 o superior. Es posible que las versiones anteriores también funcionen, pero recomendamos la 18.09 o una versión posterior.
- Python, versión 3.6 o superior.
- pip, versión 18.1 o posterior.
- AWS CLI versión 1.17.10 o posterior
  - Para instalar la versión 1 de AWS CLI, consulte Instalación de la AWS CLI versión 1.
  - Para configurar el AWS CLI, consulte Configuración de la AWS CLI.
  - Para actualizar a la versión más reciente de la versión 1 de AWS CLI, ejecute el siguiente comando.

```
pip install awscli --upgrade --user
```

> **Note**
>
> Si utiliza la instalación de MSI de la versión 1 AWS CLI en Windows, tenga en cuenta lo siguiente:
>
> - Si la instalación de la versión 1 de AWS CLI no consigue instalar botocore, intente utilizar la instalación de Python y pip.
> - Para actualizar a una versión 1 de AWS CLI posterior, deberá repetir el proceso de instalación del MSI.

- Para acceder a los recursos de Amazon Elastic Container Registry (Amazon ECR), debe conceder el siguiente permiso.
  - Amazon ECR requiere que los usuarios tengan permiso `ecr:GetAuthorizationToken` para realizar llamadas a la API AWS Identity and Access Management a través de una política de IAM antes de que puedan autenticarse en un registro, así como insertar o extraer imágenes de cualquier repositorio de Amazon ECR. Para obtener más información, consulte los ejemplos de

políticas de repositorios de Amazon ECR y el Acceso a un repositorio de Amazon ECR en la Guía del usuario de Amazon Elastic Container Registry.

Paso 1: Obtener la imagen del contenedor AWS IoT Greengrass de Amazon ECR

AWS proporciona imágenes de Docker que tienen instalado el software AWS IoT Greengrass Core.

> **Warning**
>
> A partir de la versión 1.11.6 del software AWS IoT Greengrass Core, las imágenes de Docker de Greengrass ya no incluyen Python 2.7 porque Python 2.7 llegó al final de su vida útil en 2020 y ya no recibe actualizaciones de seguridad. Si decide actualizar a estas imágenes de Docker, le recomendamos que compruebe que sus aplicaciones funcionan con las nuevas imágenes de Docker antes de implementar las actualizaciones en los dispositivos de producción. Si necesita Python 2.7 para su aplicación que usa una imagen de Docker de Greengrass, puede modificar el Dockerfile de Greengrass para incluir Python 2.7 en su aplicación.

Para ver los pasos que muestran cómo extraer la imagen de `latest` de Amazon ECR, elija su sistema operativo:

Extraer la imagen del contenedor (Linux)

Ejecute los siguientes comandos en el terminal del equipo.

1. Inicie sesión en su registro AWS IoT Greengrass en Amazon ECR.

   ```
   aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
   password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
   ```

   Si la acción se realiza correctamente, se muestra el texto `Login Succeeded` (Inicio de sesión correcto).
2. Recupere la imagen del contenedor de AWS IoT Greengrass.

   ```
   docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

> **Note**
>
> La imagen `latest` contiene la última versión estable del software AWS IoT Greengrass Core instalada en una imagen base de Amazon Linux 2. También puede extraer otras imágenes del repositorio. Para encontrar todas las imágenes disponibles, consulte la página Tags (Etiquetas) en Docker Hub o utilice el comando aws ecr list-images. Por ejemplo:
>
> ```
> aws ecr list-images --region us-west-2 --registry-id 216483018798 --
> repository-name aws-iot-greengrass
> ```

3. Habilite la protección de enlaces permanentes y simbólicos. Si está experimentando con la ejecución de AWS IoT Greengrass en un contenedor, puede habilitar la configuración solo del inicio actual.

   > **Note**
   >
   > Es posible que tenga que utilizar sudo para ejecutar estos comandos.

   - Para habilitar la configuración solo para el inicio actual:

     ```
     echo 1 > /proc/sys/fs/protected_hardlinks
     echo 1 > /proc/sys/fs/protected_symlinks
     ```

   - Para habilitar la configuración para conservarla tras los reinicios:

     ```
     echo '# AWS IoT Greengrass' >> /etc/sysctl.conf
     echo 'fs.protected_hardlinks = 1' >> /etc/sysctl.conf
     echo 'fs.protected_symlinks = 1' >> /etc/sysctl.conf

     sysctl -p
     ```

4. Habilite el reenvío de red IPv4, que es necesario para la implementación en la nube de AWS IoT Greengrass y para que las comunicaciones MQTT funcionen en Linux. En el archivo `/etc/sysctl.conf`, defina `net.ipv4.ip_forward` en 1 y, a continuación, vuelva a cargar `sysctls`.

```
sudo nano /etc/sysctl.conf
# set this net.ipv4.ip_forward = 1
sudo sysctl -p
```

> **Note**
>
> Puede utilizar el editor de su elección en lugar de nano.

Extraer la imagen del contenedor (macOS)

Ejecute los siguientes comandos en el terminal del equipo.

1. Inicie sesión en su registro AWS IoT Greengrass en Amazon ECR.

   ```
   aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
   password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
   ```

   Si la acción se realiza correctamente, se muestra el texto `Login Succeeded` (Inicio de sesión correcto).

2. Recupere la imagen del contenedor de AWS IoT Greengrass.

   ```
   docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   > **Note**
   >
   > La imagen `latest` contiene la última versión estable del software AWS IoT Greengrass Core instalada en una imagen base de Amazon Linux 2. También puede extraer otras imágenes del repositorio. Para encontrar todas las imágenes disponibles, consulte la página Tags (Etiquetas) en Docker Hub o utilice el comando aws ecr list-images. Por ejemplo:
   >
   > ```
   > aws ecr list-images --region us-west-2 --registry-id 216483018798 --
   > repository-name aws-iot-greengrass
   > ```

Extraer la imagen del contenedor (Windows)

En el símbolo del sistema, ejecute los siguientes comandos. Para poder utilizar comandos de Docker en Windows, se debe estar ejecutando Docker Desktop.

1. Inicie sesión en su registro AWS IoT Greengrass en Amazon ECR.

   ```
   aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
   password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
   ```

   Si la acción se realiza correctamente, se muestra el texto `Login Succeeded` (Inicio de sesión correcto).
2. Recupere la imagen del contenedor de AWS IoT Greengrass.

   ```
   docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   > **Note**
   >
   > La imagen `latest` contiene la última versión estable del software AWS IoT Greengrass Core instalada en una imagen base de Amazon Linux 2. También puede extraer otras imágenes del repositorio. Para encontrar todas las imágenes disponibles, consulte la página Tags (Etiquetas) en Docker Hub o utilice el comando aws ecr list-images. Por ejemplo:
   >
   > ```
   > aws ecr list-images --region us-west-2 --registry-id 216483018798 --
   > repository-name aws-iot-greengrass
   > ```

Paso 2: Crear y configurar el grupo y el núcleo de Greengrass

La imagen de Docker tiene el software de AWS IoT Greengrass Core instalado, pero debe crear un grupo y núcleo de Greengrass. Esto incluye la descarga de certificados y el archivo de configuración del núcleo.

- Siga los pasos de the section called “Módulo 2: Instalación del software de AWS IoT Greengrass Core”. Omita los pasos donde se descarga y ejecute el software de AWS IoT Greengrass Core. El software y sus dependencias de tiempo de ejecución ya están configuradas en la imagen de Docker.

Paso 3: Ejecutar AWS IoT Greengrass localmente

Una vez configurado su grupo, ya está listo para configurar e iniciar el núcleo. Para ver los pasos que muestran cómo hacerlo, elija su sistema operativo:

Ejecutar Greengrass localmente (Linux)

Ejecute los siguientes comandos en el terminal del equipo.

1. Cree una carpeta para los recursos de seguridad del dispositivo y mueva el certificado y las claves a esa carpeta. Ejecute los comandos siguientes. Sustituya la *ruta a los archivos de seguridad* por la ruta a los recursos de seguridad y sustituya CertificateID por el ID del *certificado en los nombres* de los archivos.

   ```
   mkdir /tmp/certs
   mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
   mv path-to-security-files/certificateId-public.pem.key /tmp/certs
   mv path-to-security-files/certificateId-private.pem.key /tmp/certs
   mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
   ```

2. Cree una carpeta para la configuración del dispositivo y mueva el archivo de configuración AWS IoT Greengrass principal a esa carpeta. Ejecute los comandos siguientes. Reemplace *configuration-file-path* por la ruta al archivo de configuración del agente.

   ```
   mkdir /tmp/config
   mv path-to-config-file/config.json /tmp/config
   ```

3. Inicie AWS IoT Greengrass y realice un montaje vinculado de los certificados y el archivo de configuración en el contenedor de Docker.

   Reemplace `/tmp` por la ruta donde descomprimió los certificados y el archivo de configuración.

   ```
   docker run --rm --init -it --name aws-iot-greengrass \
   --entrypoint /greengrass-entrypoint.sh \
   -v /tmp/certs:/greengrass/certs \
   -v /tmp/config:/greengrass/config \
   -p 8883:8883 \
   216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   El resultado debe tener el aspecto del siguiente ejemplo:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

Ejecutar Greengrass localmente (macOS)

Ejecute los siguientes comandos en el terminal del equipo.

1. Cree una carpeta para los recursos de seguridad del dispositivo y mueva el certificado y las claves a esa carpeta. Ejecute los comandos siguientes. Sustituya la *ruta a los archivos de seguridad* por la ruta a los recursos de seguridad y sustituya CertificateID por el ID del *certificado en los nombres* de los archivos.

   ```
   mkdir /tmp/certs
   mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
   mv path-to-security-files/certificateId-public.pem.key /tmp/certs
   mv path-to-security-files/certificateId-private.pem.key /tmp/certs
   mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
   ```

2. Cree una carpeta para la configuración del dispositivo y mueva el archivo de configuración AWS IoT Greengrass principal a esa carpeta. Ejecute los comandos siguientes. Reemplace *configuration-file-path* por la ruta al archivo de configuración del agente.

   ```
   mkdir /tmp/config
   mv path-to-config-file/config.json /tmp/config
   ```

3. Inicie AWS IoT Greengrass y realice un montaje vinculado de los certificados y el archivo de configuración en el contenedor de Docker.

   Reemplace `/tmp` por la ruta donde descomprimió los certificados y el archivo de configuración.

   ```
   docker run --rm --init -it --name aws-iot-greengrass \
   --entrypoint /greengrass-entrypoint.sh \
   -v /tmp/certs:/greengrass/certs \
   -v /tmp/config:/greengrass/config \
   -p 8883:8883 \
   216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

El resultado debe tener el aspecto del siguiente ejemplo:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

Ejecutar Greengrass localmente (Windows)

1. Cree una carpeta para los recursos de seguridad del dispositivo y mueva el certificado y las claves a esa carpeta. En el símbolo del sistema, ejecute los siguientes comandos. Sustituya la *ruta a los archivos de seguridad* por la ruta a los recursos de seguridad y sustituya CertificateID por el ID del *certificado en los nombres* de los archivos.

   ```
   mkdir C:\Users\%USERNAME%\Downloads\certs
   move path-to-security-files\certificateId-certificate.pem.crt C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\certificateId-public.pem.key C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\certificateId-private.pem.key C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\AmazonRootCA1.pem C:\Users\%USERNAME%\Downloads\certs
   ```

2. Cree una carpeta para la configuración del dispositivo y mueva el archivo de configuración AWS IoT Greengrass principal a esa carpeta. En el símbolo del sistema, ejecute los siguientes comandos. Reemplace *configuration-file-path* por la ruta al archivo de configuración del agente.

   ```
   mkdir C:\Users\%USERNAME%\Downloads\config
   move path-to-config-file\config.json C:\Users\%USERNAME%\Downloads\config
   ```

3. Inicie AWS IoT Greengrass y realice un montaje vinculado de los certificados y el archivo de configuración en el contenedor de Docker. En el símbolo del sistema, ejecute los siguientes comandos.

   ```
   docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
   entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs
   ```

```
-v c:/Users/%USERNAME%/Downloads/config:/greengrass/config -p 8883:8883
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Cuando Docker le pida que comparta su unidad `C:\` con el daemon de Docker, permita que realice un montaje vinculado del directorio `C:\` dentro del contenedor de Docker. Para obtener más información, consulte Unidades compartidas en la documentación de Docker.

El resultado debe tener el aspecto del siguiente ejemplo:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

> **Note**
>
> Si el contenedor no abre el intérprete de comandos y sale de inmediato, este problema se puede depurar realizando un montaje vinculado en los registros del tiempo de ejecución de Greengrass al iniciar la imagen. Para obtener más información, consulte the section called "Para conservar los registros del tiempo de ejecución de Greengrass fuera del contenedor Docker".

Paso 4: Configurar la creación de contenedores "Sin contenedor" para el grupo de Greengrass

Cuando se ejecuta AWS IoT Greengrass en un contenedor de Docker, todas las funciones de Lambda deben ejecutarse sin creación de contenedores. En este paso, se establece la creación de contenedores predeterminada para el grupo en No container (Sin contenedor). Debe hacerlo antes de implementar el grupo por primera vez.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija el grupo cuya configuración desea cambiar.
3. Elija la pestaña Función de Lambda.
4. En Entorno de tiempo de ejecución de funciones de Lambda predeterminado, elija Editar.

5. En el entorno de tiempo de ejecución Editar función de Lambda por defecto, en Creación de contenedores de la función de Lambda por defecto, cambie la configuración de la contenedorización.
6. Seleccione Save.

Los cambios surten efecto cuando el grupo se implementa.

Para obtener más información, consulte the section called “Configuración de la creación de contenedores predeterminada para funciones Lambda de un grupo”.

> **Note**
>
> De forma predeterminada, las funciones de Lambda utilizan la configuración de grupo para creación de contenedores. Si anula la configuración Sin contenedor de cualquier función de Lambda cuando se está ejecutando AWS IoT Greengrass en un contenedor de Docker, la implementación devuelve un error.

Paso 5: Implementar funciones de Lambda en el contenedor de Docker de AWS IoT Greengrass

Puede implementar funciones de Lambda de larga duración en el contenedor de Docker de Greengrass.

- Siga los pasos que se indican en the section called “Módulo 3 (primera parte): Funciones de Lambda en AWS IoT Greengrass” para implementar una función de Lambda Hello World de larga duración en el contenedor.

Paso 6: (Opcional) Implementar los dispositivos cliente que interactúan con la instancia de Greengrass que se ejecuta en el contenedor de Docker

También puede implementar dispositivos cliente que interactúan con AWS IoT Greengrass cuando se ejecuta en un contenedor de Docker.

- Siga los pasos que se indican en the section called “Módulo 4: Interacción con dispositivos cliente en un grupo de AWS IoT Greengrass” para implementar dispositivos cliente que se conectan al núcleo y envían mensajes MQTT.

Parar el contenedor Docker de AWS IoT Greengrass

Para parar el contenedor de Docker de AWS IoT Greengrass, pulse Ctrl+C en su terminal o en la línea de comandos. Esa acción envía `SIGTERM` al proceso de daemon de Greengrass para eliminar el proceso de daemon de Greengrass y todos los procesos de Lambda que inició el proceso del daemon. El contenedor de Docker se inicializa con proceso `/dev/init` como PID 1, lo que ayuda a eliminar los procesos zombis restantes. Para obtener más información, consulte Docker run reference.

Solución de problemas de AWS IoT Greengrass en un contenedor Docker

Utilice la siguiente información para ayudar a solucionar problemas con la ejecución de AWS IoT Greengrass en un contenedor de Docker.

Error: No se puede realizar un inicio de sesión interactivo desde un dispositivo que no sea TTY.

Solución: Este error puede producirse al ejecutar el comando `aws ecr get-login-password`. Asegúrese de haber instalado la última versión 2 o la versión 1 de AWS CLI. Le recomendamos que utilice la última versión 2 de AWS CLI. Para obtener más información, consulte Instalar AWS CLI en la Guía del usuario de AWS Command Line Interface.

Error: Unknown options: -no-include-email

Solución: Este error puede producirse al ejecutar el comando `aws ecr get-login`. Asegúrese de que tiene la última versión de la AWS CLI instalada (por ejemplo, ejecute: `pip install awscli --upgrade --user`). Si utiliza Windows e instaló la interfaz de línea de comandos (CLI) mediante el instalador MSI, debe repetir el proceso de instalación. Para obtener más información, consulte Instalación de la AWS Command Line Interface en Microsoft Windows en la Guía del usuario de AWS Command Line Interface.

Advertencia: IPv4 is disabled. Networking will not work.

Solución: Puede que reciba esta advertencia o un mensaje similar al ejecutar AWS IoT Greengrass en un equipo Linux. Habilite el enrutamiento de red IPv4 tal y como se describe en este paso. La implementación en la nube de AWS IoT Greengrass y las comunicaciones MQTT no funcionan cuando el reenvío de IPv4 no está habilitado. Para obtener más información, consulte Configure namespaced kernel parameters (sysctls) at runtime en la documentación de Docker.

Error: A firewall is blocking file Sharing between windows and the containers.

Solución: Puede que reciba este error o un mensaje `Firewall Detected` al ejecutar Docker en un equipo Windows. Esto también puede ocurrir si ha iniciado sesión en una red privada virtual (VPN) y su configuración de red impide el montaje de la unidad compartida. En esta situación, desactive la VPN y vuelva a ejecutar el contenedor Docker.

Error: An error occurred (AccessDeniedException) when calling the GetAuthorizationToken operation: User: arn:aws:iam::<account-id>:user/<user-name> is not authorized to perform: ecr:GetAuthorizationToken on resource: *

Puede recibir este error al ejecutar el comando `aws ecr get-login-password` si no tiene los permisos suficientes para acceder a un repositorio de Amazon ECR. Para obtener más información, consulte los Ejemplos de políticas de repositorios de Amazon ECR y el Acceso a un repositorio de Amazon ECR en la Guía del usuario de Amazon ECR.

Para obtener ayuda general de solución de problemas de AWS IoT Greengrass, consulte Solución de problemas.

Depuración de AWS IoT Greengrass en un contenedor Docker

Para depurar problemas con un contenedor de Docker, puede conservar los registros del tiempo de ejecución de Greengrass o asociar un intérprete de comandos interactivo al contenedor de Docker.

Para conservar los registros del tiempo de ejecución de Greengrass fuera del contenedor Docker

Puede ejecutar el contenedor de Docker de AWS IoT Greengrass después de realizar un montaje vinculado del directorio `/greengrass/ggc/var/log`. Los registros se mantendrán incluso después de que el contenedor se cierre o se elimine.

En Linux o macOS

1. Detenga todos los contenedores Docker de Greengrass que se ejecuten en el host y, a continuación, ejecute el siguiente comando en un terminal. Esto realizará un montaje vinculado en el directorio `log` de Greengrass e iniciará la imagen de Docker.

   Reemplace `/tmp` por la ruta donde descomprimió los certificados y el archivo de configuración.

   ```
   docker run --rm --init -it --name aws-iot-greengrass \
   ```

```
    --entrypoint /greengrass-entrypoint.sh \
    -v /tmp/certs:/greengrass/certs \
    -v /tmp/config:/greengrass/config \
    -v /tmp/log:/greengrass/ggc/var/log \
    -p 8883:8883 \
    216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

A continuación, puede consultar los registros en el directorio `/tmp/log` del host para ver qué sucedió mientras Greengrass estaba en ejecución en el contenedor de Docker.

En Windows

Detenga todos los contenedores Docker de Greengrass que se ejecuten en el host y, a continuación, ejecute el siguiente comando en un símbolo del sistema. Esto realizará un montaje vinculado en el directorio `log` de Greengrass e iniciará la imagen de Docker.

```
cd C:\Users\%USERNAME%\Downloads
mkdir log
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs -v c:/
Users/%USERNAME%/Downloads/config:/greengrass/config -v c:/Users/%USERNAME%/
Downloads/log:/greengrass/ggc/var/log -p 8883:8883 216483018798.dkr.ecr.us-
west-2.amazonaws.com/aws-iot-greengrass:latest
```

A continuación, puede consultar los registros en el directorio `C:/Users/%USERNAME%/Downloads/log` del host para ver qué sucedió mientras Greengrass estaba en ejecución en el contenedor de Docker.

Para asociar un intérprete de comandos interactivo a un contenedor de Docker

Puede asociar un intérprete de comandos interactivo a un contenedor de Docker de AWS IoT Greengrass que esté en ejecución. Esto le ayudará a investigar el estado del contenedor de Docker de Greengrass.

En Linux o macOS

Mientras se ejecuta el contenedor Docker de Greengrass, ejecute el siguiente comando en un terminal distinto.

```
docker exec -it $(docker ps -a -q -f "name=aws-iot-greengrass") /bin/bash
```

En Windows

Mientras se ejecuta el contenedor Docker de Greengrass, ejecute los siguientes comandos en un símbolo del sistema distinto.

```
docker ps -a -q -f "name=aws-iot-greengrass"
```

Sustituya *gg-container-id* por el resultado `container_id` del comando anterior.

```
docker exec -it gg-container-id /bin/bash
```

Acceder a recursos locales con conectores y funciones de Lambda

Esta característica está disponible para AWS IoT Greengrass Core versión 1.3 y posteriores.

Con AWS IoT Greengrass, puede crear funciones de AWS Lambda y configurar conectores en la nube e implementarlos en los dispositivos del núcleo para una ejecución local. En los núcleos Greengrass que se ejecuten con Linux, los conectores y las funciones de Lambda implementados de forma local pueden obtener acceso a recursos locales que estén físicamente presentes en el dispositivo del núcleo Greengrass. Por ejemplo, para comunicarse con dispositivos que estén conectados a través de Modbus o CANbus, puede habilitar la función de Lambda para obtener acceso al puerto serie del dispositivo del núcleo. Para configurar un acceso seguro a los recursos locales, debe garantizar la seguridad del hardware físico y el sistema operativo del dispositivo del núcleo de Greengrass.

Para empezar a obtener acceso a recursos locales, consulte los siguientes tutoriales:

- Cómo configurar el acceso a recursos locales mediante la interfaz de la línea de comando de AWS
- Cómo configurar el acceso a recursos locales mediante AWS Management Console

Tipos de recursos admitidos

Puede obtener acceso a dos tipos de recursos locales: recursos de volumen y recursos de dispositivo.

Recursos de volumen

Archivos o directorios del sistema de archivos raíz (excepto de `/sys`, `/dev` o `/var`). Entre ellas se incluyen:

- Carpetas o archivos que se utilizan para leer o escribir información en funciones de Lambda de Greengrass (por ejemplo, `/usr/lib/python2.x/site-packages/local`).
- Carpetas o archivos en el sistema de archivos /proc del host (por ejemplo, `/proc/net` o `/proc/stat`). Compatible con la versión 1.6 o posterior. Para informarse de los requisitos adicionales, consulte the section called “Recursos de volumen en el directorio /proc”.

> **Tip**
>
> Para configurar los directorios `/var`, `/var/run` y `/var/lib` como recursos de volumen, primero monte el directorio en otra carpeta y, a continuación, configure la carpeta como un recurso de volumen.

Al configurar recursos de volumen, debe especificar una ruta de origen y una ruta de destino. La ruta de origen es la ruta completa del recurso en el host. La ruta de destino es la ruta completa del recurso en el entorno del espacio de nombres de Lambda. Es el contenedor en el que se ejecuta una función o un conector de Lambda de Greengrass. Cualquier cambio que se realice en la ruta de destino se refleja en ruta de origen del sistema de archivos del host.

> **Note**
>
> Los archivos de la ruta de destino solo se pueden ver en el espacio de nombres de Lambda. No es posible verlos en un espacio de nombres normal de Linux.

Recursos de dispositivo

Archivos en `/dev`. Solo tienen permiso para obtener acceso a los recursos de dispositivo los dispositivos de caracteres o los dispositivos de bloques en `/dev`. Entre ellas se incluyen:

- Puertos de serie que se utilizan para comunicarse con dispositivos conectados a través de puertos de serie (por ejemplo, `/dev/ttyS0`, `/dev/ttyS1`).
- USB utilizados para conectar periféricos USB (por ejemplo, `/dev/ttyUSB0` o `/dev/bus/usb`).
- GPIO que se utilizan para sensores y actuadores a través de GPIO (por ejemplo, `/dev/gpiomem`).
- GPU que se utilizan para acelerar el aprendizaje automático mediante GPU a bordo (por ejemplo, `/dev/nvidia0`).
- Cámaras que se utilizan para capturar imágenes y vídeos (por ejemplo, `/dev/video0`).

> **Note**
>
> `/dev/shm` es una excepción. Se puede configurar solo como un recurso de volumen. Se debe conceder permiso `rw` a los recursos de `/dev/shm`.

AWS IoT Greengrass también admite tipos de recursos que se utilizan para llevar a cabo la inferencia de aprendizaje automático. Para obtener más información, consulte Cómo realizar la inferencia de machine learning.

Requisitos

Los siguientes requisitos se aplican para configurar el acceso seguro a los recursos locales:

- Debe utilizar el software de AWS IoT Greengrass Core versión 1.3 o posteriores. Para crear recursos para el directorio /proc del host, debe utilizar la versión 1.6 o posterior.
- Los recursos locales (incluidos los controladores y las bibliotecas necesarias) deben estar correctamente instalados en el dispositivo del núcleo de Greengrass y disponibles durante su uso.
- La operación deseada del recurso y el acceso al mismo no debe requerir privilegios raíz.
- Solo están disponibles los permisos `read` o `read and write`. Las funciones de Lambda no pueden realizar operaciones privilegiadas en los recursos.
- Debe proporcionar la ruta completa del recurso local en el sistema operativo del dispositivo de Greengrass Core.
- Un nombre o ID de recursos debe tener un máximo de 128 caracteres y utilizar el patrón `[a-zA-Z0-9:_-]+`.

Recursos de volumen en el directorio /proc

Las siguientes consideraciones se aplican a los recursos de volumen que se encuentran bajo el directorio /proc del host.

- Debe utilizar el software de AWS IoT Greengrass Core versión 1.6 o posteriores.
- Puede permitir acceso de solo lectura a las funciones de Lambda, pero no acceso de lectura y escritura. AWS IoT Greengrass administra este nivel de acceso.
- Es posible que también necesite conceder permisos de grupo de SO para habilitar el acceso de lectura en el sistema de archivos. Por ejemplo, suponga que su directorio o archivo de origen

tiene un permiso de archivo 660, lo que significa que solo el propietario o usuario del grupo tiene acceso de lectura (y escritura). En este caso, debe añadir permisos de propietario del grupo de SO al recurso. Para obtener más información, consulte the section called “Group owner file access permission (Permiso de acceso a los archivos del propietario del grupo)”.

- Tanto el entorno de host como el espacio de nombres de Lambda contienen un directorio /proc, así que asegúrese de evitar conflictos de nomenclatura cuando especifique la ruta de destino. Por ejemplo, si /proc es la ruta de origen, puede especificar /host-proc como la ruta de destino (o cualquier nombre de ruta que no sea "/proc").

Group owner file access permission (Permiso de acceso a los archivos del propietario del grupo)

Un proceso de la función de Lambda de AWS IoT Greengrass se suele ejecutar como `ggc_user` y `ggc_group`. Sin embargo, puede dar permisos de acceso a archivos adicionales al proceso de la función de Lambda en la definición de recursos locales, tal y como se indica a continuación:

- Para añadir los permisos del grupo Linux propietario del recurso, utilice el parámetro `GroupOwnerSetting#AutoAddGroupOwner` o la opción de consola Añadir automáticamente los permisos del sistema de archivos del grupo del sistema propietario del recurso.
- Para añadir los permisos de otro grupo de Linux, utilice el parámetro `GroupOwnerSetting#GroupOwner` o la opción de la consola Especifique otro grupo de sistemas para añadir los permisos del sistema. El valor `GroupOwner` se omite si `GroupOwnerSetting#AutoAddGroupOwner` es true.

Un proceso de la función de Lambda de AWS IoT Greengrass hereda todos los permisos del sistema de archivos de `ggc_user`, `ggc_group` y el grupo de Linux (si se ha añadido). Para que esta función de Lambda tenga acceso a un recurso, el proceso de función de Lambda debe tener los permisos necesarios para el recurso. Puede utilizar el comando `chmod(1)` para cambiar el permiso del recurso, si es necesario.

Véase también

- Cuotas de servicio de los recursos en la Referencia general de Amazon Web Services

Cómo configurar el acceso a recursos locales mediante la interfaz de la línea de comando de AWS

Esta característica está disponible para AWS IoT Greengrass Core versión 1.3 y posteriores.

Para utilizar un recurso local, debe añadir una definición del recurso a la definición del grupo que se ha implementado en su dispositivo del núcleo de Greengrass. La definición del grupo también debe contener la definición de una función de Lambda en la que se conceden permisos de acceso para recursos locales a las funciones de Lambda. Para obtener más información, incluidos los requisitos y las restricciones, consulte Acceder a recursos locales con conectores y funciones de Lambda.

En este tutorial, se describe el proceso para crear un recurso local y configurar el acceso a él mediante la interfaz de la línea de comandos (CLI) de AWS Command Line Interface. Para seguir los pasos del tutorial, debe haber creado un grupo de Greengrass como se describe en Empezar con AWS IoT Greengrass.

Para ver un tutorial que utilice la AWS Management Console, consulte Cómo configurar el acceso a recursos locales mediante AWS Management Console.

Creación de recursos locales

En primer lugar, utilice el comando `CreateResourceDefinition` para crear una definición de recurso que especifique los recursos a los que se accederá. En este ejemplo, se crean dos recursos, `TestDirectory` y `TestCamera`:

```
aws greengrass create-resource-definition  --cli-input-json '{
    "Name": "MyLocalVolumeResource",
    "InitialVersion": {
        "Resources": [
            {
                "Id": "data-volume",
                "Name": "TestDirectory",
                "ResourceDataContainer": {
                    "LocalVolumeResourceData": {
                        "SourcePath": "/src/LRAtest",
                        "DestinationPath": "/dest/LRAtest",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": true,
                            "GroupOwner": ""
```

```
                        }
                    }
                }
            },
            {
                "Id": "data-device",
                "Name": "TestCamera",
                "ResourceDataContainer": {
                    "LocalDeviceResourceData": {
                        "SourcePath": "/dev/video0",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": true,
                            "GroupOwner": ""
                        }
                    }
                }
            }
        ]
    }
}'
```

Recursos: una lista de objetos `Resource` del grupo de Greengrass. Un grupo de Greengrass puede tener hasta 50 recursos.

Resource#Id: el identificador único del recurso. El ID se utiliza para consultar un recurso en la configuración de la función de Lambda. Longitud máxima 128 caracteres. Patrón: [a-zA-Z0-9:_-]+.

Nombre del recurso: el nombre del recurso. El nombre del recurso se muestra en la consola de Greengrass. Longitud máxima 128 caracteres. Patrón: [a-zA-Z0-9:_-]+.

LocalDeviceResourceData# SourcePath: la ruta absoluta local del recurso del dispositivo. La ruta de origen de un recurso de dispositivo solo puede consultar un dispositivo de carácter o dispositivo de bloques bajo `/dev`.

LocalVolumeResourceData# SourcePath: La ruta absoluta local del recurso de volumen en el dispositivo principal de Greengrass. Esta ubicación se encuentra fuera del contenedor en el que se ejecuta la característica. La ruta de origen de un tipo de recurso de volumen no puede empezar con `/sys`.

LocalVolumeResourceData# DestinationPath: La ruta absoluta del recurso de volumen dentro del entorno Lambda. Esta ubicación se encuentra dentro del contenedor en el que se ejecuta la característica.

GroupOwnerSetting: le permite configurar privilegios de grupo adicionales para el proceso Lambda. Este campo es opcional. Para obtener más información, consulte Group owner file access permission (Permiso de acceso a los archivos del propietario del grupo).

GroupOwnerSetting# AutoAddGroupOwner: Si es verdadero, Greengrass añade automáticamente el propietario del recurso al grupo de sistemas operativos Linux especificado a los privilegios del proceso de Lambda. El proceso Lambda tiene los permisos de acceso a los archivos del grupo de Linux añadido.

GroupOwnerSetting# GroupOwner: Especifica el nombre del grupo de sistemas operativos Linux cuyos privilegios se agregan al proceso Lambda. Este campo es opcional.

`CreateResourceDefinition` devuelve un ARN de la versión de la definición del recurso. El ARN debe utilizarse al actualizar la definición de un grupo.

```
{
    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/resources/ab14d0b5-116e-4951-a322-9cde24a30373/versions/a4d9b882-
d025-4760-9cfe-9d4fada5390d",
    "Name": "MyLocalVolumeResource",
    "LastUpdatedTimestamp": "2017-11-15T01:18:42.153Z",
    "LatestVersion": "a4d9b882-d025-4760-9cfe-9d4fada5390d",
    "CreationTimestamp": "2017-11-15T01:18:42.153Z",
    "Id": "ab14d0b5-116e-4951-a322-9cde24a30373",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/resources/
ab14d0b5-116e-4951-a322-9cde24a30373"
}
```

Creación de la función de Greengrass

Después de crear los recursos, utilice el comando `CreateFunctionDefinition` para crear la característica de Greengrass y conceder a la característica acceso al recurso:

```
aws greengrass create-function-definition --cli-input-json '{
    "Name": "MyFunctionDefinition",
    "InitialVersion": {
        "Functions": [
            {
                "Id": "greengrassLraTest",
                "FunctionArn": "arn:aws:lambda:us-
west-2:012345678901:function:lraTest:1",
```

```
                "FunctionConfiguration": {
                    "Pinned": false,
                    "MemorySize": 16384,
                    "Timeout": 30,
                    "Environment": {
                        "ResourceAccessPolicies": [
                            {
                                "ResourceId": "data-volume",
                                "Permission": "rw"
                            },
                            {
                                "ResourceId": "data-device",
                                "Permission": "ro"
                            }
                        ],
                        "AccessSysfs": true
                    }
                }
            }
        ]
    }
}'
```

ResourceAccessPolicies: Contiene los `resourceId` y `permission` que otorgan a la función Lambda acceso al recurso. Una función de Lambda puede acceder a un máximo de 20 recursos.

ResourceAccessPolicy#Permission: Especifica qué permisos tiene la función Lambda en el recurso. Las opciones disponibles son `rw` (lectura/escritura) o `ro` (solo lectura).

AccessSysfs: Si es verdadero, el proceso Lambda puede tener acceso de lectura a la `/sys` carpeta del dispositivo principal de Greengrass. Esto se utiliza en casos en los que la función de Lambda Greengrass debe leer información del dispositivo de `/sys`.

De nuevo, `CreateFunctionDefinition` devuelve un ARN de versión de la definición de la característica. El ARN se debe utilizar en la versión de la definición del grupo.

```
{
    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad/versions/37f0d50e-ef50-4faf-
b125-ade8ed12336e",
    "Name": "MyFunctionDefinition",
    "LastUpdatedTimestamp": "2017-11-22T02:28:02.325Z",
```

```
    "LatestVersion": "37f0d50e-ef50-4faf-b125-ade8ed12336e",
    "CreationTimestamp": "2017-11-22T02:28:02.325Z",
    "Id": "3c9b1685-634f-4592-8dfd-7ae1183c28ad",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad"
}
```

Añadir la función de Lambda al grupo

Por último, utilice `CreateGroupVersion` para añadir la característica al grupo. Por ejemplo:

```
aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
--resource-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74316c/versions/31d0010f-
e19a-4c4c-8098-68b79906fb87" \
--core-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/
versions/297c419a-9deb-46dd-8ccc-341fc670138b" \
--function-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/functions/d1123830-da38-4c4c-a4b7-e92eec7b6d3e/versions/a2e90400-
caae-4ffd-b23a-db1892a33c78" \
--subscription-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b55a32fbcb6/
versions/470c858c-7eb3-4abd-9d48-230236bfbf6a"
```

> **Note**
>
> Para obtener información sobre cómo obtener el ID del grupo para utilizarlo con este comando, consulte the section called “Obtener el ID del grupo”.

Se devuelve una nueva versión del grupo:

```
{
    "Arn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/groups/
b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5/versions/291917fb-ec54-4895-823e-27b52da25481",
    "Version": "291917fb-ec54-4895-823e-27b52da25481",
    "CreationTimestamp": "2017-11-22T01:47:22.487Z",
    "Id": "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5"
}
```

Su grupo de Greengrass ahora contiene la función LRATest Lambda que tiene acceso a dos recursos: y. TestDirectory TestCamera

Este ejemplo de función de Lambda `lraTest.py`, escrito en Python, escribe en el recurso del volumen local:

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

Estos comandos los proporciona la API de Greengrass para crear y gestionar definiciones del recurso y versiones de definiciones del recurso:

- CreateResourceDefinition

- CreateResourceDefinitionVersion
- DeleteResourceDefinition
- GetResourceDefinition
- GetResourceDefinitionVersion
- ListResourceDefinitions
- ListResourceDefinitionVersions
- UpdateResourceDefinition

Solución de problemas

- P: ¿Por qué la implementación de mi grupo de Greengrass falla con un error similar a:

  ```
  group config is invalid:
      ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
  ```

  R: Este error indica que el proceso Lambda no tiene permiso para acceder a los recursos especificados. La solución es cambiar el permiso a los archivos del recurso para que Lambda pueda acceder a él. (Consulte Group owner file access permission (Permiso de acceso a los archivos del propietario del grupo) para obtener información).

- P: Cuando configuro `/var/run` como un recurso de volumen, ¿por qué la función de Lambda falla al iniciar con un mensaje de error en runtime.log?

  ```
  [ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
   container.
  container_linux.go:259: starting container process caused "process_linux.go:345:
  container init caused \"rootfs_linux.go:62: mounting \\\"/var/run\\\" to rootfs \\\"/
  greengrass/ggc/packages/1.3.0/rootfs_sys\\\" at \\\"/greengrass/ggc/packages/1.3.0/
  rootfs_sys/run\\\"
  caused \\\"invalid argument\\\"\""
  ```

  R: Actualmente, AWS IoT Greengrass core no es compatible con la configuración de `/var`, `/var/run` y `/var/lib` como recursos de volumen. Una alternativa es montar `/var`, `/var/run` o `/var/lib` en otra carpeta primero y, a continuación, configurar la carpeta como un recurso de volumen.

- P: Cuando configuro `/dev/shm` como un recurso de volumen con un permiso de solo lectura, ¿por qué la función de Lambda no se inicia y da un error en runtime.log?:

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
 container.
container_linux.go:259: starting container process caused "process_linux.go:345:
container init caused \"rootfs_linux.go:62: mounting \\\"/dev/shm\\\" to rootfs \\\"/
greengrass/ggc/packages/1.3.0/rootfs_sys\\\" at \\\"/greengrass/ggc/packages/1.3.0/
rootfs_sys/dev/shm\\\"
caused \\\"operation not permitted\\\"\""”
```

R: `/dev/shm` solo se puede configurar como de lectura o escritura. Cambie el permiso en el nivel de recursos a `rw` para resolver el problema.

Cómo configurar el acceso a recursos locales mediante AWS Management Console

Esta característica está disponible para AWS IoT Greengrass Core versión 1.3 y posteriores.

Puede configurar funciones de Lambda para obtener acceso seguro a los recursos locales en el dispositivo host del núcleo de Greengrass. Los recursos locales son los buses y periféricos que se encuentran físicamente en el mismo host o volúmenes del sistema de archivos del sistema operativo del host. Para obtener más información, incluidos los requisitos y las restricciones, consulte Acceder a recursos locales con conectores y funciones de Lambda.

Este tutorial describe cómo utilizar la AWS Management Console para configurar el acceso a los recursos locales que se encuentran en un dispositivo del núcleo de AWS IoT Greengrass. Contiene los siguientes pasos de alto nivel:

1. Creación de un paquete de implementación de la función de Lambda
2. Crear y publicar una función de Lambda.
3. Añadir la función de Lambda al grupo
4. Agregar un recurso local al grupo
5. Agregar suscripciones al grupo
6. Implementar el grupo

Para ver un tutorial que utilice la AWS Command Line Interface, consulte Cómo configurar el acceso a recursos locales mediante la interfaz de la línea de comando de AWS.

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Un grupo de Greengrass y un núcleo de Greengrass (versión 1.3 o posterior). Para crear un grupo o dispositivo del núcleo de Greengrass, consulte Empezar con AWS IoT Greengrass.
- Los siguientes directorios están en el dispositivo del núcleo de Greengrass:
  - /src/LRAtest
  - /dest/LRAtest

  El grupo propietario de estos directorios debe tener acceso de lectura y escritura a los directorios. Puede utilizar el siguiente comando para conceder el acceso:

```
sudo chmod 0775 /src/LRAtest
```

Paso 1: Crear un paquete de implementación de la función de Lambda

En este paso, va a crear un paquete de implementación de la función de Lambda, que es un archivo ZIP que contiene el código de la característica y dependencias. También puede descargar AWS IoT Greengrass Core SDK para que se incluya en el paquete como una dependencia.

1. En su equipo, copie el siguiente script de Python en un archivo local denominado `lraTest.py`. Contiene la lógica de la aplicación para la función de Lambda.

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)
```

```
# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass
 Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

2. Desde la página de descargas del Core SDK AWS IoT Greengrass, descargue el SDK AWS IoT Greengrass básico para Python en su ordenador.
3. Descomprima el paquete descargado para obtener el SDK. El SDK es la carpeta `greengrasssdk`.
4. Comprima los siguientes elementos en un archivo denominado `lraTestLambda.zip`:

   - `lraTest.py`. Lógica de la aplicación.
   - `greengrasssdk`. Biblioteca necesaria para todas las funciones de Lambda de Python.

   El archivo `lraTestLambda.zip` es el paquete de implementación de la función de Lambda. Ya está listo para crear una función de Lambda y cargar el paquete de implementación.

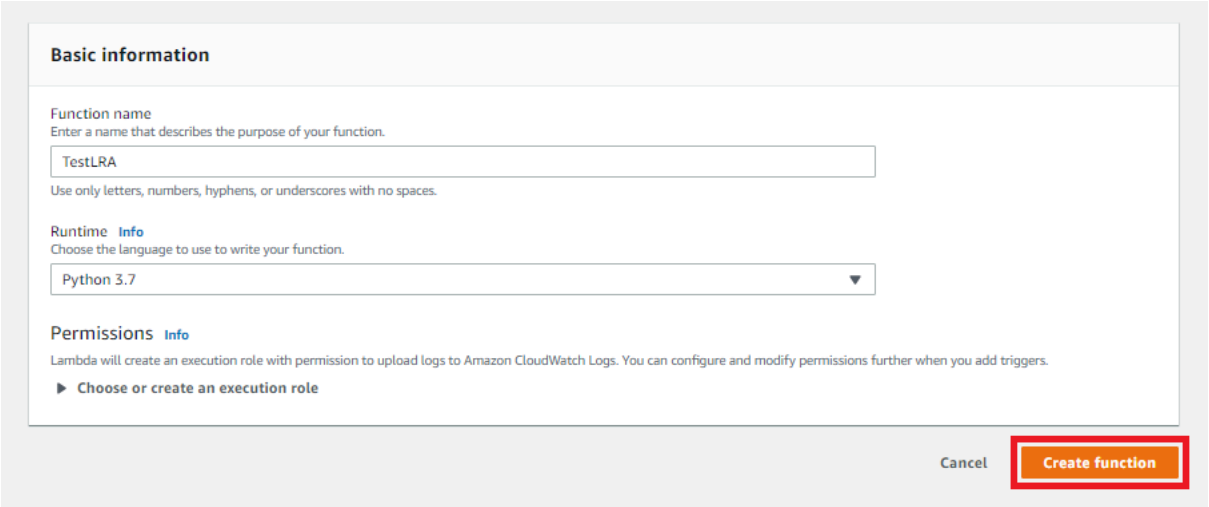
Paso 2: Crear y publicar una función de Lambda

En este paso, va a utilizar la consola de AWS Lambda para crear una función de Lambda y va a configurarla para utilizar el paquete de implementación. A continuación, publicará una versión de la característica y creará un alias.

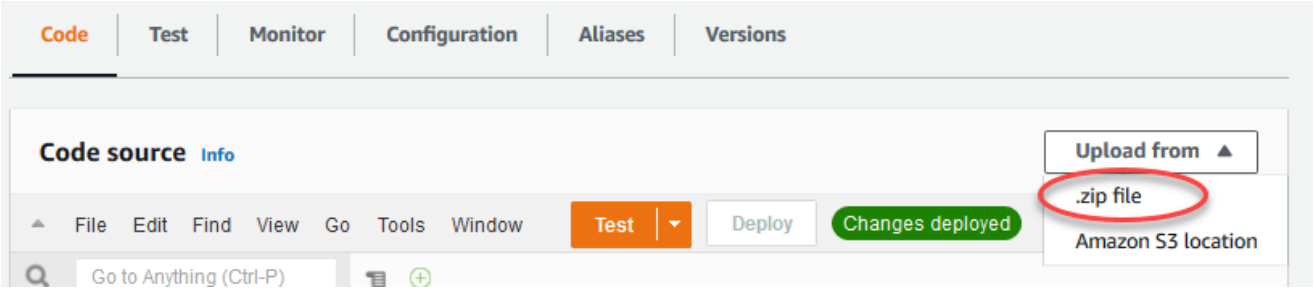
Primero, cree una función de Lambda.

1. En la AWS Management Console, elija Servicios y abra la consola de AWS Lambda.

2. Elija Funciones.
3. Elija Crear función, y, a continuación, elija Autor desde cero.
4. En la sección Basic information (Información básica), especifique los siguientes valores.
   a. En Nombre de la función, introduzca **`TestLRA`**.
   b. En Runtime (Tiempo de ejecución), elija Python 3.7.
   c. En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol.
5. Elija Crear función.



6. Cargue el paquete de implementación de la función de Lambda y registre el controlador.
   a. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip.



   b. Seleccione Cargar y, a continuación, elija su paquete de implementación `lraTestLambda.zip`. A continuación, elija Guardar.

c. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

   - En Runtime (Tiempo de ejecución), elija Python 3.7.
   - En Handler (Controlador), escriba lraTest.function_handler.

d. Seleccione Save.

> **Note**
>
> El botón de prueba de la consola de AWS Lambda no funciona con esta función. El SDK AWS IoT Greengrass Core no contiene los módulos necesarios para ejecutar las funciones de Lambda de Greengrass de forma independiente en la consola AWS Lambda. Estos módulos (por ejemplo, `greengrass_common`) se suministran a las funciones una vez desplegados en el núcleo de Greengrass.

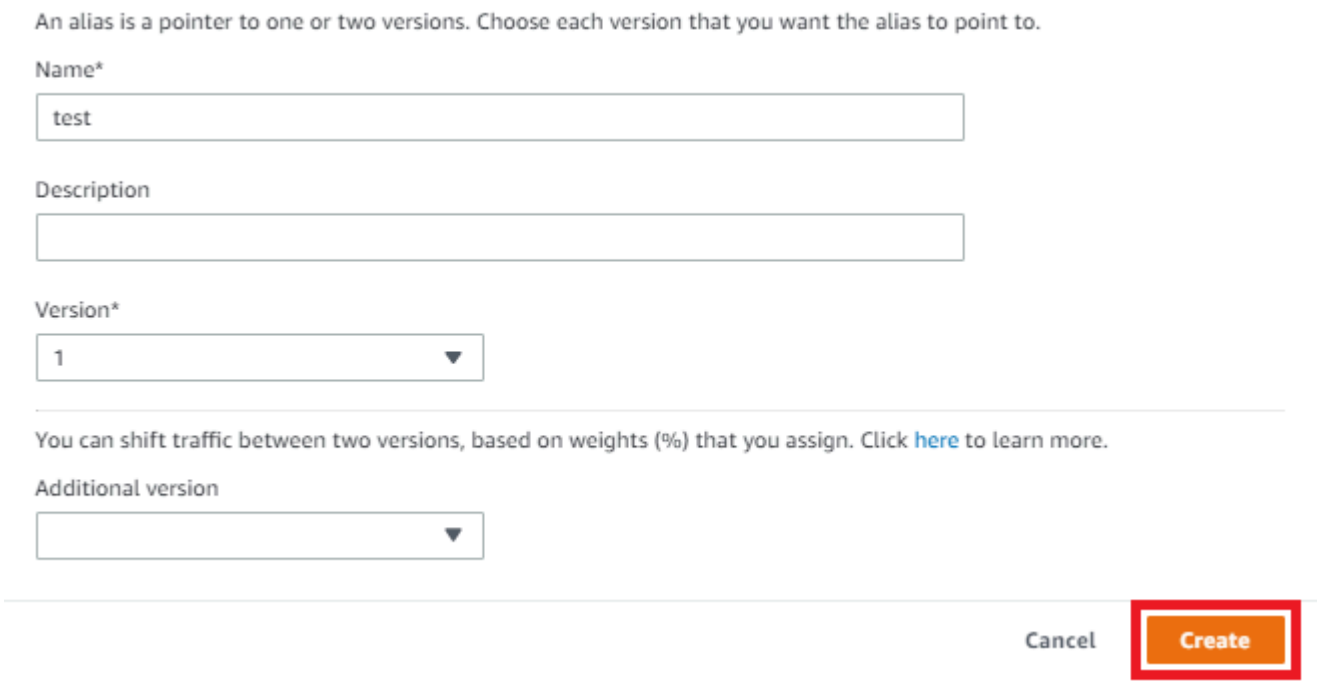
A continuación, publique la primera versión de la función de Lambda. A continuación, cree un alias para la versión.

Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

7. En Actions (Acciones), elija Publish new version (Publicar nueva versión).
8. En Version description (Descripción de versión), escriba **`First version`** y, a continuación, elija Publish (Publicar).
9. En la página de configuración de TestLRA: 1, en el menú Actions (Acciones), elija Create alias (Crear alias).
10. En la página Crear un alias, para Nombre, introduzca **`test`**. En Version (Versión), escriba 1.

> **Note**
>
> AWS IoT Greengrass no admite alias de Lambda para versiones de $LATEST.

11. Seleccione Create (Crear).



Ahora puede añadir la función de Lambda al grupo de Greengrass.

Paso 3: Añadir la función de Lambda al grupo de Greengrass

En este paso, va a añadir la función al grupo y va a configurar el ciclo de vida de la función.

Primero, añada función de Lambda al grupo de Greengrass.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija el grupo de Greengrass en el que desea añadir la función de Lambda.
3. En la página de configuración del grupo, elija la pestaña Funciones de Lambda.
4. En la sección Mis funciones de Lambda, seleccione Añadir.
5. En la página Añadir función de Lambda, elija la Función de lambda. Seleccionar **`TestLRA`**.
6. Elija la versión de la función de Lambda.
7. En la sección Configuración de la función de Lambda, seleccione Usuario y grupo del sistema y Creación de contenedores de la función de Lambda.

A continuación, configure el ciclo de vida de la función de Lambda.

8. En Timeout (Tiempo de espera), elija 30 seconds (30 segundos).

> **Important**
>
> Las funciones de Lambda que utilizan recursos locales (tal y como se describe en este procedimiento) deben ejecutarse en un contenedor de Greengrass. De lo contrario, si intenta implementar la función, se producirá un error en la implementación. Para obtener más información, consulte Containerization (Creación de contenedores).

9. En la parte inferior de la página, elija Agregar función de Lambda.

Paso 4: Agregar un recurso local al grupo de Greengrass

En este paso, va a añadir un recurso de volumen local al grupo de Greengrass y va a conceder a la característica acceso de lectura y escritura al recurso. Un recurso local tiene un ámbito de nivel de grupo. Puede conceder permisos para que cualquier función de Lambda del grupo obtenga acceso al recurso.

1. En la página de configuración del grupo, elija la pestaña Recursos.
2. En la sección Recursos locales, seleccione Añadir.
3. En la página Añadir un recurso local, utilice los siguientes valores.

   a. En Nombre del recurso, escriba **`testDirectory`**.

   b. En Resource type (Tipo de recurso), elija Device (Dispositivo).

   c. En Ruta del dispositivo, escriba **`/src/LRAtest`**. Esta ruta debe existir en el sistema operativo del host.

      La ruta del dispositivo local es la ruta absoluta local del recurso en el sistema de archivos del dispositivo del núcleo. Esta ubicación se encuentra fuera del contenedor en el que se ejecuta la característica. La ruta no puede comenzar con `/sys`.

   d. En Destination path (Ruta de destino), escriba **`/dest/LRAtest`**. Esta ruta debe existir en el sistema operativo del host.

      El valor de ruta de destino es la ruta completa del recurso en el espacio de nombres de Lambda. Esta ubicación se encuentra dentro del contenedor en el que se ejecuta la característica.

e. En Propietario del grupo del sistema y permiso de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.

   La opción Propietario del grupo del sistema y permiso de acceso a archivos le permite conceder permisos adicionales de acceso a archivos al proceso Lambda. Para obtener más información, consulte Group owner file access permission (Permiso de acceso a los archivos del propietario del grupo).

4. Seleccione Add resource (Añadir recurso). La página Resources muestra el nuevo recurso testDirectory.

Paso 5: Agregar suscripciones al grupo de Greengrass

En este paso, va a añadir dos suscripciones al grupo de Greengrass. Estas suscripciones permiten la comunicación bidireccional entre la función de Lambda e AWS IoT.

En primer lugar, cree una suscripción para que la función de Lambda envíe mensajes a AWS IoT.

1. En la página de configuración del grupo, elija la pestaña Suscripciones.
2. Elija Add (Agregar).
3. En la página Crear una suscripción, configure el origen y el destino de la siguiente manera:
   a. Para Tipo de origen, elija Función de lambda y, a continuación, TestLRA.
   b. Para Tipo de destino, elija Servicio y, a continuación, Nube de IoT.
   c. En Filtro por temas, introduzca **`LRA/test`** y, a continuación, seleccione Crear suscripción.
4. La página Subscriptions muestra la nueva suscripción.

   A continuación, configure una suscripción que invoque la característica desde AWS IoT.
5. En la página Subscriptions, elija Add Subscription.
6. En la página Select your source and target, configure el origen y el destino, de la siguiente manera:
   a. En Tipo de origen, elija la función de Lambda y, a continuación, elija Nube de IoT.
   b. Para Tipo de destino, elija Servicio y, a continuación, Nube de IoT.

c. Elija Next (Siguiente).

7. En la página Filter your data with a topic (Filtrar los datos con un tema), en el campo Topic filter (Filtro de temas), escriba **`invoke/LRAFunction`** y, a continuación, elija Next (Siguiente).
8. Elija Finalizar. La página Subscriptions muestra ambas suscripciones.

Paso 6: Implementar el grupo de AWS IoT Greengrass

En este paso, va a implementar la versión actual de la definición del grupo.

1. Asegúrese de que el núcleo de AWS IoT Greengrass se está ejecutando. Ejecute los siguientes comandos en el terminal de Raspberry Pi según sea necesario.

   a. Para comprobar si el daemon está en ejecución:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, el daemon está en ejecución.

   > **Note**
   >
   > La versión que figura en la ruta depende de la versión del software AWS IoT Greengrass Core que esté instalada en el dispositivo del núcleo.

   b. Iniciar el daemon:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. En la página de configuración de grupo, elija Implementar.

   > **Note**
   >
   > La implementación produce un error si ejecuta la función de Lambda sin creación de contenedores e intenta acceder a los recursos locales asociados.

3. Si se le solicita, en la pestaña Función de lambda, en Funciones de Lambda del sistema, seleccione Detector IP y, a continuación, Editar y, a continuación, Detectar automáticamente.

Esto permite a los dispositivos adquirir automáticamente la información de conexión del dispositivo principal, como la dirección IP, el DNS y el número de puerto. Se recomienda la detección automática, pero AWS IoT Greengrass también es compatible con puntos de conexión especificados manualmente. Solo se le solicitará el método de detección la primera vez que se implemente el grupo.

> **Note**
>
> Si se le solicita, conceda permiso para crear el rol de servicio de Greengrass y asócielo a su Cuenta de AWS en el Región de AWS actual. Este rol permite a AWS IoT Greengrass acceder a los servicios de AWS.

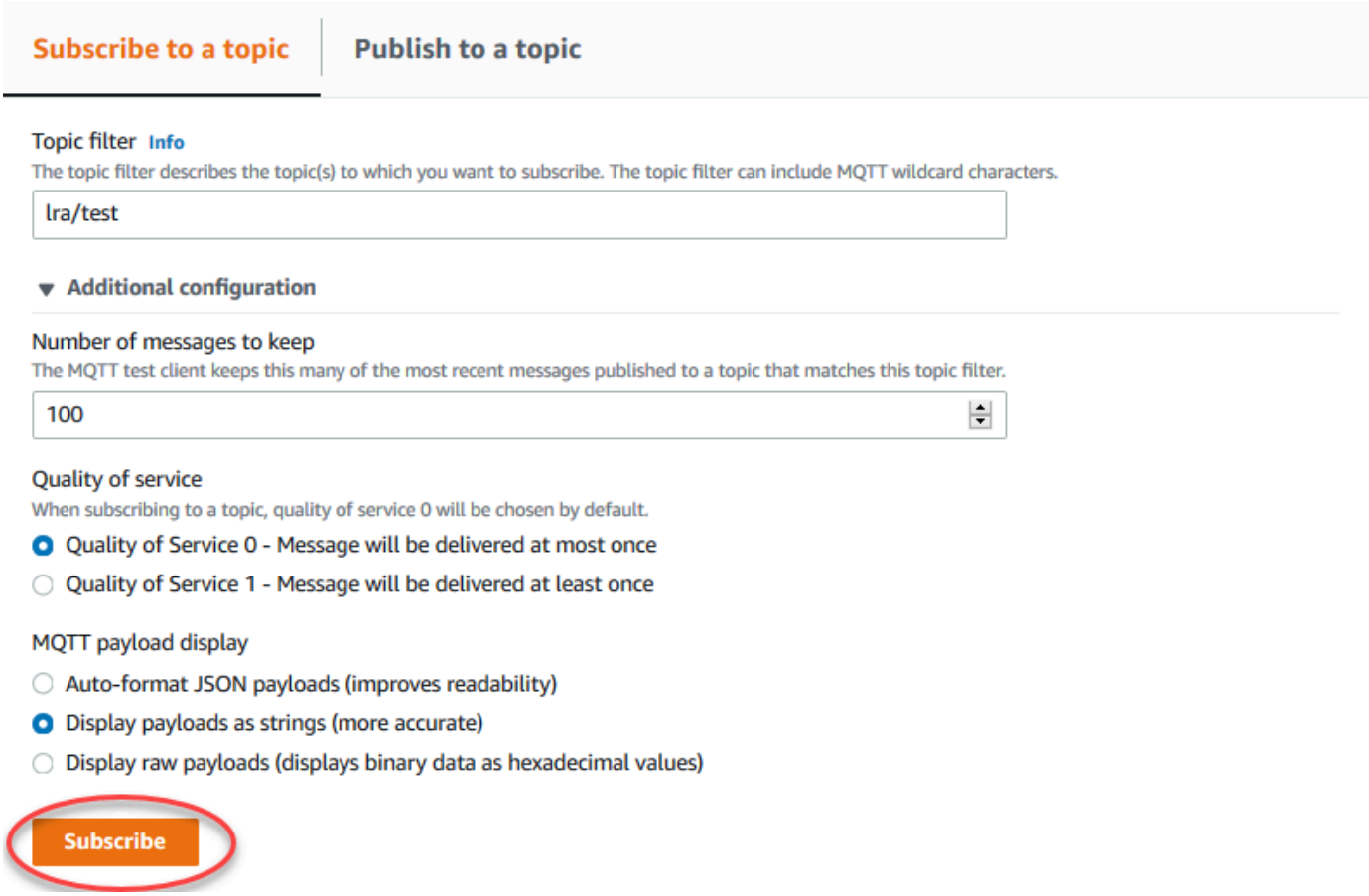
En la página Deployments (Implementaciones), se muestra la marca temporal, el ID de versión y el estado de la implementación. Al terminar, el estado de la implementación es Completado.

Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

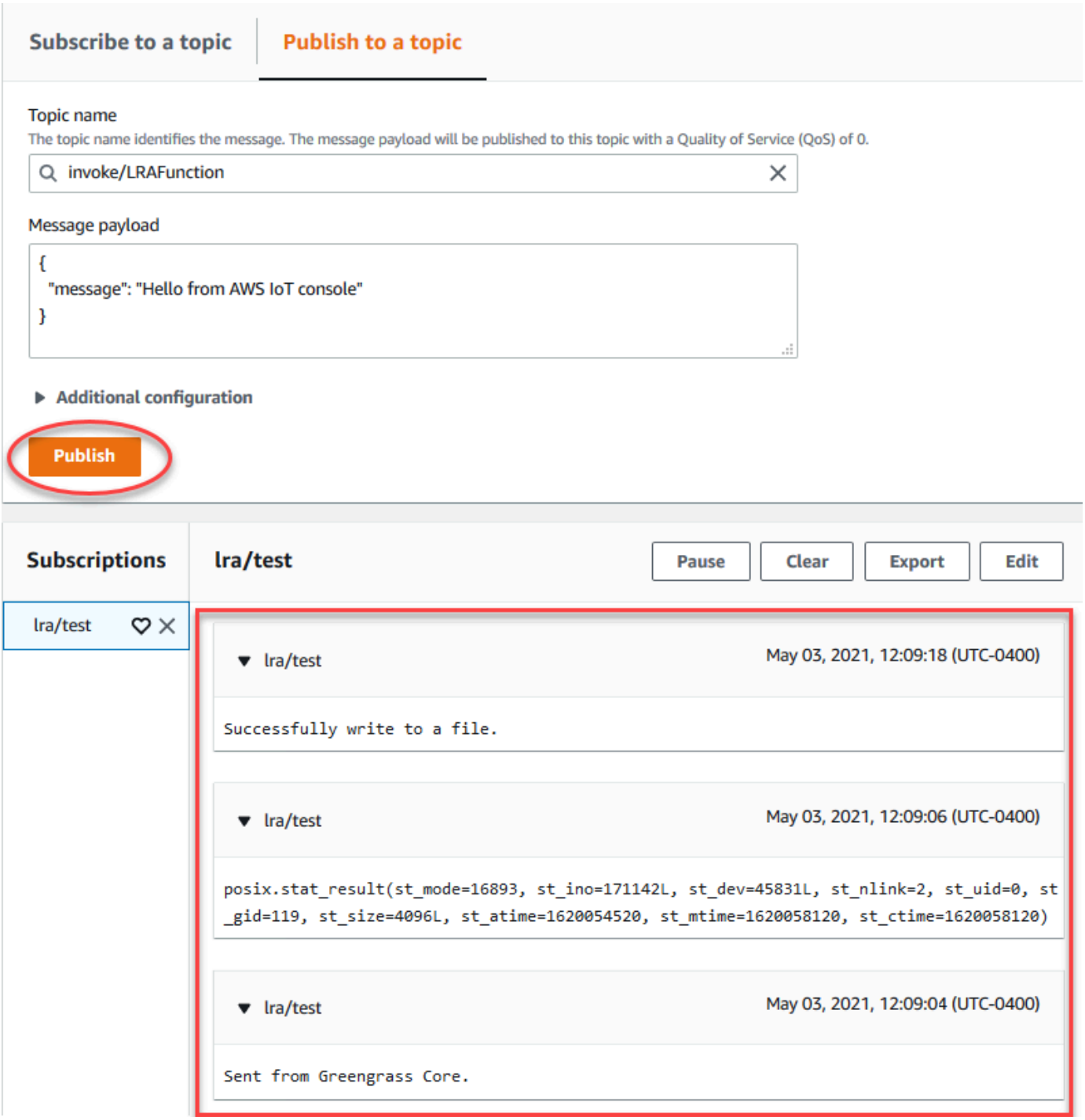
Probar el acceso al recurso local

Ahora puede verificar si el acceso al recurso local se ha configurado correctamente. Para probarlo, suscríbase al tema `LRA/test` y publique en el tema `invoke/LRAFunction`. La prueba se realiza correctamente si la función de Lambda envía la carga esperada a AWS IoT.

1. En el menú de navegación de la consola AWS IoT, en Probar, elija el cliente de prueba MQTT.
2. En Suscribirse a un tema, en Filtro de temas, introduzca **`LRA/test`**.
3. En Información adicional, en Visualización de cargas útiles en MQTT, seleccione Mostrar cargas útiles como cadenas.
4. Elija Subscribe. La función de Lambda realiza la publicación en el tema LRA/test.



5. En Publicar en un tema, en el nombre del tema**`invoke/LRAFunction`**, escriba y, a continuación, elija Publicar para invocar la función de Lambda. La prueba se realiza correctamente si la página muestra las tres cargas de mensajes de la característica.



El archivo de prueba creado por la función de Lambda se encuentra en el directorio `/src/LRAtest` del dispositivo del núcleo de Greengrass. Aunque la función de Lambda escribe en un archivo del directorio `/dest/LRAtest`, ese archivo está visible solo en el espacio de nombres de Lambda. No puede verlo en un espacio de nombres normal de Linux. Cualquier cambio que se realice en la ruta de destino se refleja en ruta de origen del sistema de archivos.

Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

Cómo realizar la inferencia de machine learning

Esta característica está disponible para AWS IoT Greengrass Core versión 1.6 o posterior.

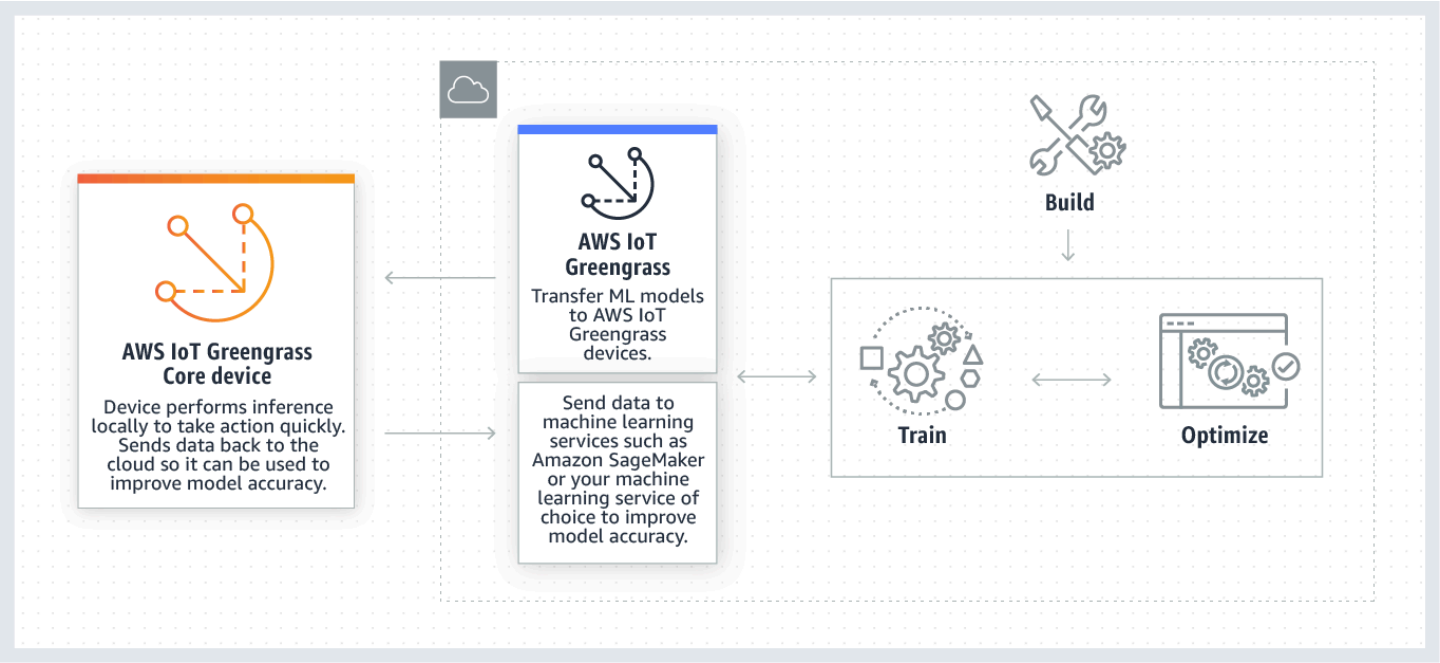
AWS IoT Greengrass le permite realizar la inferencia de machine learning (ML) en la periferia con datos generados localmente utilizando modelos entrenados en la nube. Benefíciese de la baja latencia y el ahorro de costos que supone la ejecución de inferencias locales, aprovechando al mismo tiempo la potencia de cómputo de la nube para el entrenamiento de modelos y el procesamiento complejo.

Para comenzar a realizar la inferencia local, consulte the section called “Cómo configurar la inferencia de machine learning”.

Funcionamiento de la inferencia de machine learning de AWS IoT Greengrass

Puede entrenar sus modelos de inferencia en cualquier lugar, desplegarlos localmente como recursos de machine learning en un grupo de Greengrass y luego acceder a ellos desde las funciones de Lambda de Greengrass. Por ejemplo, puede crear y entrenar modelos de aprendizaje profundo en SageMaker e implementarlos en el núcleo de Greengrass. A continuación, las funciones de Lambda pueden utilizar los modelos locales para realizar inferencias con los dispositivos cliente y enviar datos de entrenamiento nuevos de vuelta a la nube.

En el siguiente diagrama se muestra el flujo de trabajo de la inferencia de machine learning de AWS IoT Greengrass.



La inferencia de machine learning de AWS IoT Greengrass simplifica todos los pasos del flujo de trabajo de machine learning, incluidos:

- La creación e implementación de prototipos de marco de trabajo de machine learning.
- El acceso a modelos entrenados en la nube y su implementación en dispositivos del núcleo de Greengrass.
- La creación de aplicaciones de inferencia que pueden obtener acceso a aceleradores de hardware (como GPU y FPGA) como recursos locales.

Recursos de machine learning

Los recursos de machine learning representan modelos de inferencia entrenados en la nube que se implementan en un AWS IoT Greengrass. Para implementar recursos de machine learning, primero debe añadirlos a un grupo de Greengrass y, a continuación, definir cómo podrán obtener acceso a ellos las funciones de Lambda de dicho grupo. Durante la implementación del grupo, AWS IoT Greengrass recupera los paquetes de modelos de origen de la nube y los extrae en directorios dentro del espacio de nombres del tiempo de ejecución de Lambda. A continuación, las funciones de Lambda de Greengrass utilizan los modelos implementados localmente para llevar a cabo la inferencia.

Para actualizar un modelo implementado localmente, primero debe actualizar el modelo de origen (en la nube) correspondiente al recurso de machine learning y, a continuación, implementar el grupo.

Durante la implementación, AWS IoT Greengrass comprueba si existen cambios en el origen. Si se detectan cambios, AWS IoT Greengrass actualiza el modelo local.

Orígenes de modelos admitidos

AWS IoT Greengrass admite orígenes de modelos de SageMaker y Amazon S3 para los recursos de machine learning.

Los siguientes requisitos se aplican a los orígenes de modelos:

- Los buckets de S3 que almacenan los orígenes de modelos de SageMaker y Amazon S3 no deben estar cifrados mediante SSE-C. En los buckets que utilizan cifrado del lado del servidor, la inferencia de machine learning de AWS IoT Greengrass solo admite actualmente opciones de cifrado SSE-S3 o SSE-KMS. Para obtener más información sobre las opciones de cifrado del lado del servidor, consulte Protección de datos con el cifrado del lado del servidor en la Guía del usuario de Amazon Simple Storage Service.
- Los nombres de los buckets de S3 que almacenan los orígenes de modelos de SageMaker y Amazon S3 no deben incluir puntos (`.`). Para obtener más información, consulte la regla sobre el uso de buckets de estilo alojado virtuales con SSL en Reglas para la nomenclatura de buckets en la Guía del usuario de Amazon Simple Storage Service.
- El soporte de Región de AWS de nivel de servicio debe estar disponible para AWS IoT Greengrass y SageMaker. Actualmente, AWS IoT Greengrass admite modelos de SageMaker en las siguientes regiones:
  - Este de EE. UU. (Ohio)
  - Este de EE. UU. (Norte de Virginia)
  - Oeste de EE. UU. (Oregón)
  - Asia Pacific (Mumbai)
  - Asia Pacific (Seoul)
  - Asia Pacífico (Singapur)
  - Asia Pacífico (Sídney)
  - Asia-Pacífico (Tokio)
  - Europa (Fráncfort)
  - Europa (Irlanda)
  - Europa (Londres)

- AWS IoT Greengrass debe tener permisos `read` en el origen de modelos, como se describe en las secciones siguientes.

SageMaker

AWS IoT Greengrass es compatible con los modelos que se guardan como trabajos de entrenamiento de SageMaker. SageMaker es un servicio de machine learning totalmente administrado que puede usar para crear y entrenar modelos mediante algoritmos integrados o personalizados. Para obtener más información, consulte ¿Qué es SageMaker? en la Guía para desarrolladores de SageMaker.

Si ha configurado el entorno de SageMaker creando un bucket cuyo nombre contiene `sagemaker`, entonces AWS IoT Greengrass tiene permisos suficientes para obtener acceso a los trabajos de entrenamiento de SageMaker. La política administrada de `AWSGreengrassResourceAccessRolePolicy` permite el acceso a los buckets cuyo nombre contiene la cadena `sagemaker`. Esta política está asociada al rol de servicio de Greengrass.

En caso contrario, debe conceder a AWS IoT Greengrass permisos `read` en el bucket en el que se almacena el trabajo de entrenamiento. Para ello, integre la siguiente política insertada en el rol de servicio. Puede incluir varios ARN de bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-name"
            ]
        }
    ]
}
```

Amazon S3

AWS IoT Greengrass es compatible con modelos que se almacenan en Amazon S3 como archivos `tar.gz` o `.zip`.

Si desea que AWS IoT Greengrass pueda obtener acceso a los modelos que se almacenan en buckets de Amazon S3, debe conceder a AWS IoT Greengrass `read` permisos; para ello, realice una de las acciones siguientes:

- Almacene el modelo en un bucket cuyo nombre contenga `greengrass`.

  La política administrada de `AWSGreengrassResourceAccessRolePolicy` permite el acceso a los buckets cuyo nombre contiene la cadena `greengrass`. Esta política está asociada al rol de servicio de Greengrass.

- Incruste una política insertada en el rol de servicio de Greengrass.

  Si el nombre del bucket no contiene `greengrass`, añada la siguiente política insertada al rol de servicio. Puede incluir varios ARN de bucket.

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "s3:GetObject"
              ],
              "Resource": [
                  "arn:aws:s3:::my-bucket-name"
              ]
          }
      ]
  }
  ```

  Para obtener más información, consulte Integración de políticas insertadas en la Guía de usuario de IAM.

Requisitos

Los siguientes requisitos se aplican para crear y utilizar recursos de machine learning:

- Debe utilizar Núcleo de AWS IoT Greengrass versión 1.6. o posterior.

- Las funciones de Lambda definidas por el usuario pueden realizar operaciones `read` o `read and write` en el recurso. Los permisos para otras operaciones no están disponibles. El modo de creación de contenedores de las funciones de Lambda afiliadas determina cómo configura los permisos de acceso. Para obtener más información, consulte the section called “Acceso a recursos de aprendizaje automático”.
- Debe proporcionar la ruta completa del recurso en el sistema operativo del dispositivo del núcleo.
- Un nombre o ID de recursos debe tener un máximo de 128 caracteres y utilizar el patrón `[a-zA-Z0-9:_-]+`.

Entornos de ejecución y bibliotecas para la inferencia de machine learning

Puede utilizar los siguientes entornos de tiempos de ejecución y bibliotecas de machine learning con AWS IoT Greengrass.

- Entorno de ejecución de aprendizaje profundo de Amazon SageMaker Neo
- Apache MXNet
- TensorFlow

Estas bibliotecas y entornos de ejecución precompilados se pueden instalar en plataformas NVIDIA Jetson TX2, Intel Atom y Raspberry Pi. Para obtener información sobre la descarga, consulte the section called “Bibliotecas y entornos de ejecución de aprendizaje automático compatibles”. Puede instalarlos directamente en el dispositivo central.

Asegúrese de leer la siguiente información sobre la compatibilidad y las limitaciones.

Tiempo de ejecución del aprendizaje profundo de SageMaker Neo

Puede utilizar el tiempo de ejecución del aprendizaje profundo de SageMaker Neo para realizar la inferencia con modelos de machine learning optimizados en sus dispositivos de AWS IoT Greengrass. Estos modelos están optimizados con el compilador de aprendizaje profundo de SageMaker Neo para mejorar las velocidades de predicción de la inferencia del machine learning. Para obtener más información acerca de la optimización de modelos en SageMaker, consulte la documentación de SageMaker Neo.

> **Note**
>
> Actualmente, puede optimizar los modelos de machine learning utilizando el compilador de aprendizaje profundo Neo únicamente en regiones específicas de Amazon Web Services. Sin embargo, puede utilizar el tiempo de ejecución del aprendizaje profundo Neo con modelos optimizados en cada una de las Región de AWS donde se admite el núcleo AWS IoT Greengrass. Para obtener más información, consulte cómo configurar la inferencia de machine learning optimizado.

Control de versiones de MXNet

Apache MXNet no garantiza actualmente la compatibilidad con versiones posteriores, por lo que es posible que los modelos que entrene utilizando versiones posteriores del marco de trabajo no funcionen correctamente en versiones anteriores de dicho marco de trabajo. Para evitar conflictos entre el entrenamiento de modelos y las etapas de distribución de modelos, así como para proporcionar una experiencia integral uniforme, utilice la misma versión del marco de trabajo MXNet en ambas etapas.

MXNet en Raspberry Pi

Las funciones de Lambda de Greengrass que obtienen acceso a los modelos de MXNet locales deben definir la siguiente variable de entorno:

```
MXNET_ENGINE_TYPE=NativeEngine
```

Puede definir la variable de entorno en el código de la característica o agregarla a la configuración específica del grupo de la característica. Para ver un ejemplo acerca de cómo agregarla como una opción de configuración, consulte este paso.

> **Note**
>
> Para hacer un uso general del marco MXNet (por ejemplo, ejecutar un ejemplo de código de terceros), la variable de entorno debe estar configurada en Raspberry Pi.

Limitaciones de la distribución de modelos de TensorFlow en Raspberry Pi

Las siguientes recomendaciones para mejorar los resultados de la inferencia se basan en las pruebas que hemos realizado con las bibliotecas precompiladas de ARM de 32 bits en la plataforma Raspberry Pi. Estas recomendaciones van dirigidas a usuarios avanzados únicamente a modo de referencia, sin garantías de ningún tipo.

- Los modelos que se entrenan con el formato de punto de comprobación deberían "congelarse" en el formato del búfer del protocolo antes de su distribución. Para ver un ejemplo, consulte la biblioteca de modelos de clasificación de imágenes de TensorFlow-Slim.
- No utilice las bibliotecas TF-Estimator y TF-Slim en el código de inferencia o de entrenamiento. En su lugar, utilice el patrón de archivo de carga de modelos `.pb` que se muestra en el ejemplo siguiente.

```
graph = tf.Graph()
graph_def = tf.GraphDef()
graph_def.ParseFromString(pb_file.read())
with graph.as_default():
  tf.import_graph_def(graph_def)
```

> **Note**
>
> Para obtener más información sobre las plataformas compatibles con TensorFlow, consulte Installing TensorFlow en la documentación de TensorFlow.

Acceso a recursos de machine learning desde funciones

Las funciones de Lambda definidas por el usuario pueden acceder a los recursos de machine learning para ejecutar inferencia local en el núcleo de AWS IoT Greengrass.. Un recurso de aprendizaje automático consiste en el modelo entrenado y otros artefactos que se descargan en el dispositivo principal.

Para permitir que una función de Lambda tenga acceso a un recurso de machine learning en el núcleo, debe adjuntar el recurso a la función de Lambda y definir los permisos de acceso. El modo de creación de contenedores de la función de Lambda asociada (o adjunta) determina cómo hacerlo.

Permisos de acceso para recursos de aprendizaje automático

A partir de AWS IoT Greengrass Core v1.10.0, puede definir un propietario de recurso para un recurso de aprendizaje automático. El propietario del recurso representa el grupo del sistema operativo y los permisos que AWS IoT Greengrass utiliza para descargar los artefactos de recursos. Si no se define el propietario de un recurso, los artefactos de recursos descargados sólo son accesibles para raíz.

- Si las funciones de Lambda que no están en un contenedor acceden a un recurso de machine learning, debe definir un propietario de recurso porque no hay control de permisos desde el contenedor. Las funciones de Lambda que no están en un contenedor pueden heredar permisos de propietario de recursos y utilizarlos para acceder al recurso.

- Si solo las funciones de Lambda en contenedor acceden al recurso, se recomienda utilizar permisos de nivel de función en lugar de definir un propietario de recurso.

Propiedades del propietario del recurso

Un propietario de recurso especifica un propietario de grupo y permisos de propietario de grupo.

Propietario del grupo. El ID del grupo (GID) de un grupo de SO Linux existente en el dispositivo principal. Los permisos del grupo se agregan al proceso de Lambda. Específicamente, el GID se agrega a los identificadores de grupo suplementarios de la función de Lambda.

Si una función de Lambda del grupo Greengrass está configurada para ejecutarse como el mismo grupo de SO que el propietario del recurso para un recurso de machine learning, el recurso debe estar asociado a la función de Lambda. De lo contrario, la implementación falla porque esta configuración proporciona permisos implícitos que la función de Lambda puede utilizar para acceder al recurso sin autorización AWS IoT Greengrass. La comprobación de validación de implementación se omite si la función de Lambda se ejecuta como raíz (UID=0).

Le recomendamos que utilice un grupo de SO que no sea utilizado por otros recursos, funciones de Lambda o archivos en el núcleo de Greengrass. El uso de un grupo de SO compartido proporciona a las funciones de Lambda asociadas más permisos de acceso de los que necesitan. Si utiliza un grupo de SO compartido, también debe adjuntarse una función de Lambda a todos

los recursos de machine learning que utilizan el grupo de SO compartido. De lo contrario, se produce un error en la implementación.

Permisos de propietario del grupo. El permiso de sólo lectura o lectura y escritura que se agregará al proceso de Lambda.

Las funciones de Lambda que no están en un contenedor deben heredar estos permisos de acceso al recurso. Las funciones de Lambda en contenedor pueden heredar estos permisos de nivel de recursos o definir permisos de nivel de función. Si definen permisos de nivel de función, los permisos deben ser igual o más restrictivos que los permisos de nivel de recursos.

En la tabla siguiente se muestran las configuraciones de permisos de acceso admitidas.

GGC v1.10 or later

Propiedad	Si solo las funciones de Lambda en contenedor acceden al recurso	Si alguna función de Lambda que no está en un contenedor accede al recurso
Propiedades de nivel de función		
Permisos (lectura/escritura)	Necesario a menos que el recurso defina un propietario de recurso. Si se define un propietario de recurso, los permisos de nivel de función deben ser igual o más restrictivos que los permisos de propietario de recurso. Si solo las funciones de Lambda en contenedor acceden al recurso, se recomienda que no defina un propietario de recurso.	Funciones de Lambda que no están en un contenedor: No admitido. Las funciones de Lambda que no están en un contenedor deben heredar permisos de nivel de recursos. Funciones de Lambda en contenedores: Opcional, pero debe ser igual o más restrictivo que los permisos de nivel de recursos.
Propiedades de nivel de recursos		

Propiedad	Si solo las funciones de Lambda en contenedor acceden al recurso	Si alguna función de Lambda que no está en un contenedor accede al recurso
Propietario del recurso	Opcional (no recomendado).	Obligatorio.
Permisos (lectura/escritura)	Opcional (no recomendado).	Obligatorio.

GGC v1.9 or earlier

Propiedad	Si solo las funciones de Lambda en contenedor acceden al recurso	Si alguna función de Lambda que no está en un contenedor accede al recurso
Propiedades de nivel de función		
Permisos (lectura/escritura)	Obligatorio.	No admitido.
Propiedades de nivel de recursos		
Propietario del recurso	No admitido.	No admitido.
Permisos (lectura/escritura)	No admitido.	No admitido.

> **Note**
>
> Cuando se utiliza la API AWS IoT Greengrass para configurar funciones y recursos de Lambda, también se requiere la propiedad `ResourceId` de nivel de función. La propiedad `ResourceId` adjunta el recurso de machine learning a la función de Lambda.

Definición de permisos de acceso para funciones de Lambda (consola)

En la consola de AWS IoT, se definen los permisos de acceso al configurar un recurso de machine learning o adjuntar uno a una función de Lambda.

Funciones de Lambda en contenedores

Si solo se adjuntan funciones de Lambda en contenedor al recurso de machine learning.

- Elija Ningún grupo de sistemas como propietario del recurso para el recurso de machine learning. Esta es la configuración recomendada cuando solo las funciones de Lambda en contenedor acceden al recurso de machine learning De lo contrario, podría conceder a las funciones Lambda adjuntas más permisos de acceso de los que necesitan.

Funciones de Lambda no organizadas en contenedores (requiere GGC v 1.10 o posterior)

Si se adjunta alguna función de Lambda que no está en un contenedor al recurso de machine learning:

- Especifique el ID de grupo del sistema (GID) que se va a utilizar como propietario del recurso para el recurso de machine learning. Elija Especificar grupo de sistemas y permisos e introduzca el GID. Puede utilizar el comando `getent group` en el dispositivo principal para buscar el ID de un grupo de sistemas..

- Elija Acceso de sólo lectura o Acceso de lectura y escritura para los permisos del grupo de sistemas.

Definición de permisos de acceso para funciones de Lambda (API)

En la API AWS IoT Greengrass, se definen permisos para los recursos de machine learning en la propiedad `ResourceAccessPolicy` de la función de Lambda o la propiedad `OwnerSetting` del recurso.

Funciones de Lambda en contenedores

Si solo se adjuntan funciones de Lambda en contenedor al recurso de machine learning.

- Para las funciones de Lambda en contenedores, defina los permisos de acceso en la propiedad `Permission` de la propiedad `ResourceAccessPolicies`. Por ejemplo:

```
"Functions": [
    {
        "Id": "my-containerized-function",
```

```
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id",
                        "Permission": "ro-or-rw"
                    }
                ]
            },
            "MemorySize": 512,
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- Para los recursos de aprendizaje automático, omita la propiedad `OwnerSetting`. Por ejemplo:

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package"
            }
        }
    }
]
```

Esta es la configuración recomendada cuando solo las funciones de Lambda en contenedores acceden al recurso de machine learning. De lo contrario, podría conceder a las funciones Lambda adjuntas más permisos de acceso de los que necesitan.

Funciones de Lambda no organizadas en contenedores (requiere GGC v 1.10 o posterior)

Si se adjunta alguna función de Lambda que no está en un contenedor al recurso de machine learning:

- Para funciones de función de Lambda que no están en un contenedor, omita la propiedad `Permission` en `ResourceAccessPolicies`. Esta configuración es necesaria y permite que la función herede el permiso de nivel de recursos. Por ejemplo:

```
"Functions": [
    {
        "Id": "my-non-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "Execution": {
                    "IsolationMode": "NoContainer",
                },
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id"
                    }
                ]
            },
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- Para las funciones de Lambda en contenedores que también tienen acceso al recurso de machine learning, omita la propiedad `Permission` en `ResourceAccessPolicies` o defina un permiso que sea igual o más restrictivo que el permiso de nivel de recursos. Por ejemplo:

```
"Functions": [
    {
        "Id": "my-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id",
                        "Permission": "ro-or-rw" // Optional, but cannot exceed
 the GroupPermission defined for the resource.
```

```
                    }
                ]
            },
            "MemorySize": 512,
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- Para los recursos de aprendizaje automático, defina la propiedad `OwnerSetting`, incluidas las propiedades secundarias `GroupOwner` y `GroupPermission`. Por ejemplo:

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package",
                "OwnerSetting": {
                    "GroupOwner": "os-group-id",
                    "GroupPermission": "ro-or-rw"
                }
            }
        }
    }
]
```

Acceso a recursos de machine learning desde el código de la función de Lambda

Las funciones de Lambda definidas por el usuario utilizan interfaces de SO específicas de la plataforma para acceder a recursos de machine learning en un dispositivo principal.

GGC v1.10 or later

Para las funciones de Lambda en contenedor, el recurso se monta dentro del contenedor Greengrass y está disponible en la ruta de destino local definida para el recurso. Para las funciones de Lambda que no están en un contenedor, el recurso se vincula a un directorio de

trabajo específico de Lambda y se pasa a la variable de entorno `AWS_GG_RESOURCE_PREFIX` en el proceso de Lambda.

Para obtener la ruta de acceso a los artefactos descargados de un recurso de machine learning, las funciones de Lambda anexan la variable de entorno `AWS_GG_RESOURCE_PREFIX` a la ruta de destino local definida para el recurso. Para las funciones de Lambda en contenedor, el valor devuelto es una sola barra diagonal (`/`).

```
resourcePath = os.getenv("AWS_GG_RESOURCE_PREFIX") + "/destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

GGC v1.9 or earlier

Los artefactos descargados de un recurso de aprendizaje automático se encuentran en la ruta de destino local definida para el recurso. Solo las funciones Lambda en contenedores pueden acceder a los recursos de machine learning en AWS IoT Greengrass Core versión 1.9 y versiones anteriores.

```
resourcePath = "/local-destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

Su implementación de carga del modelo depende de su biblioteca ML.

Solución de problemas

Utilice la siguiente información para solucionar problemas relacionados con el acceso a recursos de aprendizaje automático.

Temas

- invalidMLModelOwner : GroupOwnerSetting se proporciona en el recurso del modelo ML, pero GroupOwner o no GroupPermission está presente
- NoContainer la función no puede configurar el permiso al adjuntar recursos de Machine Learning. <function-arn>hace referencia a un recurso de aprendizaje automático <resource-id>con permiso <ro/rw> en la política de acceso a los recursos.
- <function-arn>La función se refiere a un recurso de Machine Learning al que <resource-id>le falta permiso tanto en uno como ResourceAccessPolicy en el recurso OwnerSetting.

- <function-arn>La función hace referencia al recurso Machine Learning <resource-id>con el permiso\ "rw\», mientras que la configuración del propietario del recurso GroupPermission solo permite\ "ro\».
- NoContainer La función <function-arn>hace referencia a los recursos de la ruta de destino anidada.
- Lambda <function-arn> obtiene acceso al recurso <resource-id> al compartir el mismo ID de propietario del grupo

invalidMLModelOwner : GroupOwnerSetting se proporciona en el recurso del modelo ML, pero GroupOwner o no GroupPermission está presente

Solución: recibe este error si un recurso de aprendizaje automático contiene el ResourceDownloadOwnerSettingobjeto pero la `GroupPermission` propiedad requerida `GroupOwner` o no está definida. Para resolver este problema, defina la propiedad que falta.

NoContainer la función no puede configurar el permiso al adjuntar recursos de Machine Learning. <function-arn>hace referencia a un recurso de aprendizaje automático <resource-id>con permiso <ro/rw> en la política de acceso a los recursos.

Solución: recibirá este error si una función de Lambda que no está en un contenedor especifica permisos de nivel de característica para un recurso de machine learning. Las funciones que no están en un contenedor deben heredar permisos de los permisos de propietario de recursos definidos en el recurso de machine learning. Para resolver este problema, elija heredar permisos de propietario de recursos (consola) o quitar los permisos de la política de acceso a recursos (API) de la función de Lambda.

<function-arn>La función se refiere a un recurso de Machine Learning al que <resource-id>le falta permiso tanto en uno como ResourceAccessPolicy en el recurso OwnerSetting.

Solución: recibirá este error si los permisos para el recurso de machine learning no están configurados para la función de Lambda adjunta o el recurso. Para resolver este problema, configure los permisos en la ResourceAccessPolicypropiedad de la función Lambda o en la OwnerSettingpropiedad del recurso.

<function-arn>La función hace referencia al recurso Machine Learning <resource-id>con el permiso\ "rw\», mientras que la configuración del propietario del recurso GroupPermission solo permite\ "ro\».

Solución: recibirá este error si los permisos de acceso definidos para la función de Lambda adjunta superan los permisos de propietario de recursos definidos para el recurso de machine learning. Para resolver este problema, establezca permisos más restrictivos para la función de Lambda o permisos menos restrictivos para el propietario del recurso.

NoContainer La función <function-arn>hace referencia a los recursos de la ruta de destino anidada.

Solución: recibirá este error si varios recursos de machine learning conectados a una función de Lambda no contenedora utilizan la misma ruta de destino o una ruta de destino anidada. Para resolver este problema, especifique rutas de destino separadas para los recursos.

Lambda <function-arn> obtiene acceso al recurso <resource-id> al compartir el mismo ID de propietario del grupo

Solución: recibirá este error `runtime.log` si se especifica el mismo grupo de sistemas operativos como la identidad Ejecutar como de la función de Lambda y el propietario del recurso de machine learning, pero el recurso no está adjunto a la función de Lambda. Esta configuración da a la función de Lambda permisos implícitos que puede utilizar para acceder al recurso sin autorización de AWS IoT Greengrass.

Para resolver este problema, utilice un grupo de SO diferente para una de las propiedades o adjunte el recurso de machine learning a la función de Lambda.

Véase también

- Cómo realizar la inferencia de machine learning
- the section called “Cómo configurar la inferencia de machine learning”
- the section called “Cómo configurar la inferencia de machine learning optimizado”

- Referencia de la API de AWS IoT Greengrass Version 1

Cómo configurar la inferencia de machine learning mediante AWS Management Console

Para seguir los pasos de este tutorial, necesita Núcleo de AWS IoT Greengrass versión 1.10 o posterior.

Puede realizar la inferencia de machine learning (ML) localmente en un dispositivo central de Greengrass utilizando datos generados localmente. Para obtener información, incluidos los requisitos y las restricciones, consulte Cómo realizar la inferencia de machine learning.

En este tutorial se describe cómo utilizar la AWS Management Console para configurar un grupo de Greengrass para ejecutar una aplicación de inferencias de Lambda que reconoce imágenes de una cámara localmente, sin enviar datos a la nube. La aplicación de inferencias obtiene acceso al módulo de la cámara de un dispositivo Raspberry Pi y ejecuta la inferencia mediante el modelo SqueezeNet de código abierto.

El tutorial contiene los siguientes pasos generales:

1. Configuración de Raspberry Pi
2. Instalar el marco MXNet
3. Crear un paquete de modelo
4. Crear y publicar una función de Lambda.
5. Añadir la función de Lambda al grupo
6. Agregar recursos al grupo
7. Agregar una suscripción al grupo
8. Implementar el grupo
9. Probar la aplicación

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Raspberry Pi 4 modelo B o Raspberry Pi 3 modelo B/B+, configurados y configurados para su uso con AWS IoT Greengrass. Para configurar su Raspberry Pi con AWS IoT Greengrass, ejecute el

script de configuración del dispositivo Greengrass o asegúrese de haber completado el módulo 1 y el módulo 2 de Empezar con AWS IoT Greengrass.

> **Note**
>
> Es posible que la Raspberry Pi requiera una fuente de alimentación de 2,5 A para ejecutar los marcos de aprendizaje profundo que se utilizan normalmente para la clasificación de imágenes. Una fuente de alimentación con una potencia inferior podría provocar el reinicio del dispositivo.

- Módulo de cámara Raspberry Pi V2 de 8 megapíxeles, 1080p. Para obtener información sobre cómo configurar la cámara, consulte Conectar la cámara en la documentación de Raspberry Pi.
- Un grupo de Greengrass y un núcleo de Greengrass. Para obtener información acerca de cómo crear un núcleo o grupo de Greengrass, consulte Empezar con AWS IoT Greengrass.

> **Note**
>
> En este tutorial se utiliza un dispositivo Raspberry Pi, pero AWS IoT Greengrass es compatible con otras plataformas, como Intel Atom y NVIDIA Jetson TX2. En el ejemplo de Jetson TX2, pueden utilizarse imágenes estáticas en lugar de imágenes transmitidas desde una cámara. Si usa el ejemplo de Jetson TX2, es posible que necesite instalar Python 3.6 en lugar de Python 3.7. Si necesita más información acerca de cómo configurar el dispositivo para poder instalar el software AWS IoT Greengrass Core, consulte the section called “Configuración de otros dispositivos”.
>
> Para las plataformas de terceros que AWS IoT Greengrass no son compatibles, debe ejecutar la función de Lambda en modo no contenerizado. Para ejecutarla en modo no contenerizado, debe ejecutar la función de Lambda como raíz. Para obtener más información, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.” y the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

Paso 1: Configurar el Raspberry Pi

En este paso, va a instalar actualizaciones del sistema operativo Raspbian, el software del módulo de cámara y las dependencias de Python, y va a habilitar la interfaz de la cámara.

Ejecute los siguientes comandos en el terminal de Raspberry Pi.

1. Instale las actualizaciones en Raspbian.

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

2. Instale la interfaz `picamera` del módulo de cámara y las demás bibliotecas de Python que sean necesarias para este tutorial.

   ```
   sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
   ```

   Valide la instalación:

   - Asegúrese de que la instalación de Python 3.7 incluye pip.

     ```
     python3 -m pip
     ```

     Si pip no está instalado, descárguelo del sitio web de pip y ejecute el siguiente comando.

     ```
     python3 get-pip.py
     ```

   - Asegúrese de que la versión de Python es 3.7 o superior.

     ```
     python3 --version
     ```

     Si en la salida aparece una versión anterior, ejecute el siguiente comando.

     ```
     sudo apt-get install -y python3.7-dev
     ```

   - Asegúrese de que Setuptools y Picamera se han instalado correctamente.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
     sudo -u ggc_user bash -c 'python3 -c "import picamera"'
     ```

     Si el resultado no contiene errores, la validación es correcta.

> **Note**
>
> Si el ejecutable de Python instalado en el dispositivo es `python3.7`, utilice `python3.7` en lugar de `python3` con los comandos de este tutorial. Asegúrese de que la instalación de pip corresponde a la versión `python3` o `python3.7` correcta para evitar errores de dependencia.

3. Reinicie el Raspberry Pi.

```
sudo reboot
```

4. Abra la herramienta de configuración de Raspberry Pi.

```
sudo raspi-config
```

5. Utilice las teclas de flecha para abrir Interfacing Options (Opciones de interfaz) y habilitar la interfaz de la cámara. Si se le solicita, permita que el dispositivo se reinicie.
6. Utilice el siguiente comando para probar la configuración de la cámara.

```
raspistill -v -o test.jpg
```

Se abre una ventana de vista previa en el Raspberry Pi, se guarda una imagen denominada `test.jpg` en el directorio actual y se muestra información sobre la cámara en el terminal de Raspberry Pi.

Paso 2: Instalar el marco MXNet

En este paso, va a instalar bibliotecas de MXNet en su Raspberry Pi.

1. Inicie sesión en su Raspberry Pi de forma remota.

```
ssh pi@your-device-ip-address
```

2. Abra la documentación de MXNet, diríjase a Installing MXNet y siga las instrucciones para instalar MXNet en el dispositivo.

> **Note**
>
> Recomendamos instalar la versión 1.5.0 y compilar MXNet desde el código fuente para este tutorial a fin de evitar conflictos entre dispositivos.

3. Después de instalar MXNet, valide la siguiente configuración:

   - Asegúrese de que la cuenta `ggc_user` del sistema puede utilizar el marco MXNet.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
     ```

   - Compruebe que NumPy esté instalado.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import numpy"'
     ```

Paso 3: Crear un paquete de modelo de MXNet

En este paso, cree un paquete de modelo que contenga un modelo MXNet preentrenado de muestra para cargarlo en Amazon Simple Storage Service (Amazon S3). AWS IoT Greengrass puede utilizar un paquete de modelo de Amazon S3, siempre que utilice el formato tar.gz o zip.

1. En su equipo, descargue el ejemplo de MXNet para Raspberry Pi de the section called "Ejemplos de aprendizaje automático".
2. Descomprima el archivo `mxnet-py3-armv7l.tar.gz` descargado.
3. Vaya al directorio `squeezenet`.

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
   ```

   El archivo `squeezenet.zip` de este directorio es el paquete de modelo. Contiene artefactos de código abierto SqueezeNet para un modelo de clasificación de imágenes. Más adelante, cargará este paquete de modelo en Amazon S3.

Paso 4: Crear y publicar una función de Lambda

En este paso, va a crear un paquete de implementación de funciones de Lambda y una función de Lambda. A continuación, publicará una versión de la función y creará un alias.

Primero, cree el paquete de implementación de funciones de Lambda.

1. En el equipo, desplácese hasta el directorio `examples` del paquete de ejemplo en el que descomprimió the section called “Crear un paquete de modelo”.

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/examples
   ```

   El directorio `examples` contiene el código de la función y las dependencias.

   - `greengrassObjectClassification.py` es el código de inferencia que se utiliza en este tutorial. Puede usar este código como plantilla para crear su propia función de inferencia.
   - `greengrasssdk` es la versión 1.5.0 del AWS IoT Greengrass Core SDK para Python.

   > **Note**
   >
   > Si hay una nueva versión disponible, puede descargarla y actualizar la versión del SDK del paquete de implementación. Para obtener más información, AWS IoT Greengrassvea SDK de Python en GitHub.

2. Comprima el contenido del directorio `examples` en un archivo llamado `greengrassObjectClassification.zip`. Este es el paquete de implementación.

   ```
   zip -r greengrassObjectClassification.zip .
   ```

   > **Note**
   >
   > Asegúrese de que los archivos `.py` y las dependencias se encuentran en la raíz del directorio.

Ahora, cree la función de Lambda.

3. En la consola AWS IoT, seleccione Funciones y Crear función.
4. Elija Autor desde cero y utilice los siguientes valores para crear su función:

   - En Nombre de la característica, introduzca **`greengrassObjectClassification`**.

- En Runtime (Tiempo de ejecución), elija Python 3.7.

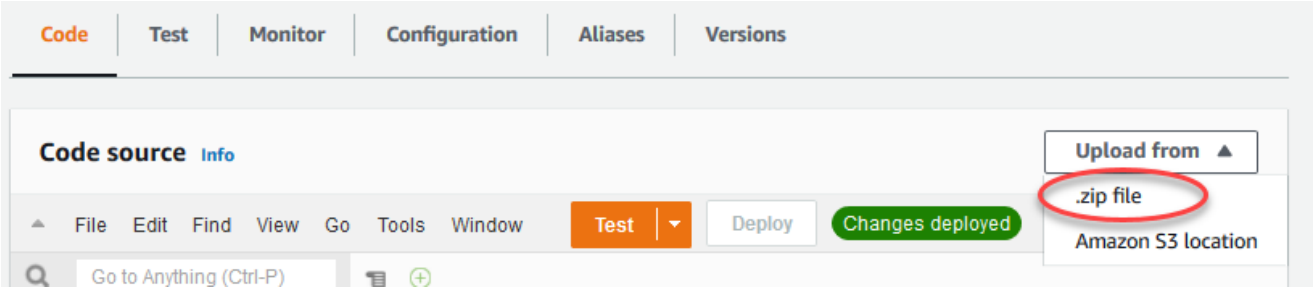
En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol.

5. Elija Crear Función.

Ahora, cargue el paquete de implementación de la función de Lambda y registre el controlador.

6. Elija su función de Lambda y cargue su paquete de implementación de funciones de Lambda.

   a. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip..



   b. Seleccione Cargar y, a continuación, elija su paquete de implementación `greengrassObjectClassification.zip`. A continuación, elija Guardar.

   c. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

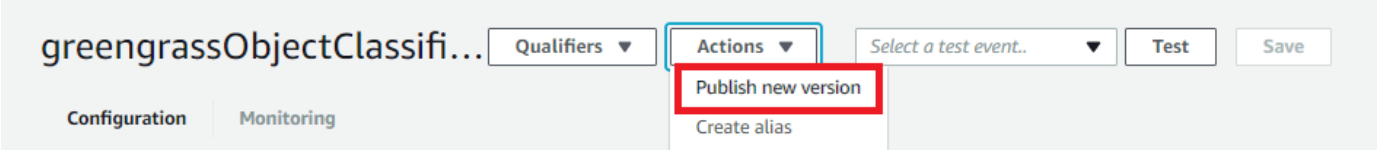
      - En Runtime (Tiempo de ejecución), elija Python 3.7.
      - En Handler (Controlador), escriba **`greengrassObjectClassification.function_handler`**.

      Seleccione Save.

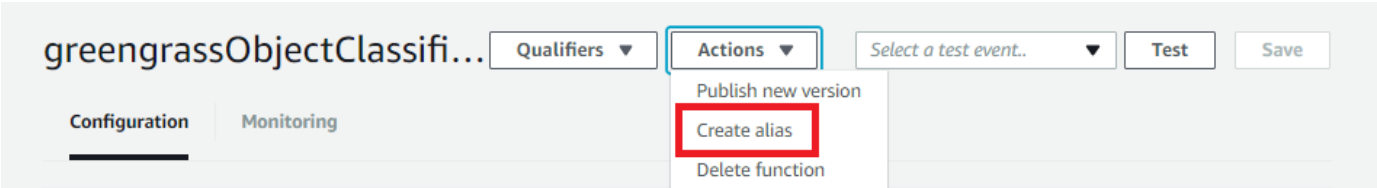
A continuación, publique la primera versión de la función de Lambda. A continuación, cree un alias para la versión.

> **Note**
>
> Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

7. En el menú Actions, elija Publish new version.



8. En Version description (Descripción de versión), escriba **`First version`** y, a continuación, elija Publish (Publicar).
9. En la página de configuración de greengrassObjectClassification: 1, en el menú Actions (Acciones), elija Create alias (Crear alias).



10. En la página Create a new alias, utilice los valores siguientes:
    - En Name (Nombre), ingrese **`mlTest`**.
    - En Version (Versión), escriba **`1`**.

> **Note**
>
> AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST.

11. Seleccione Save.

Ahora, añada función de Lambda al grupo de Greengrass.

Paso 5: Añadir la función de Lambda al grupo de Greengrass

En este paso, va a agregar la función de Lambda al grupo y a configurar su ciclo de vida y las variables de entorno.

Primero, añada función de Lambda al grupo de Greengrass.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. En la página de configuración del grupo, seleccione la pestaña Funciones de Lambda.
3. En la sección Mis funciones de Lambda, seleccione Añadir.
4. Para la función de Lambda, seleccione greengrassObjectClassification.
5. Para la versión de la función de Lambda, elija Alias:MLTest.

A continuación, configure el ciclo de vida y las variables de entorno de la función de Lambda.

6. En la sección Configuración de la función de Lambda, realice las siguientes actualizaciones.

> **Note**
>
> Le recomendamos que ejecute la función de Lambda sin creación de contenedores, a menos que su modelo de negocio lo requiera. Esto permite el acceso a la GPU y la cámara del dispositivo sin necesidad de configurar los recursos del dispositivo. Si ejecuta sin creación de contenedores, también debe conceder acceso raíz a las funciones de Lambda AWS IoT Greengrass.

   a. Para ejecutar sin creación de contenedores:

      - En Usuario y grupo del sistema, elija **`Another user ID/group ID`**. En ID de usuario del sistema, introduzca **`0`**. Para el ID de grupo del sistema, introduzca **`0`**.

        Esto permite que la función de Lambda se ejecute como raíz. Para obtener más información sobre cómo ejecutar un trabajo, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

> **Tip**
>
> También debe actualizar el archivo `config.json` para conceder acceso raíz a la función de Lambda. Para informarse sobre este procedimiento, consulte the section called “Ejecución de una función de Lambda como raíz”.

- Para la creación de contenedores de la función de Lambda, elija Sin contenedor.

  Para obtener más información sobre la ejecución sin creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.
- En Tiempo de espera, escriba **`10 seconds`**.
- En Ancladas, elija Verdadero

  Para obtener más información, consulte the section called “Configuración del ciclo de vida”.

b. Para ejecutarlo en modo contenerizado, en su lugar:

> **Note**
>
> No recomendamos ejecutarlo en modo contenerizado a menos que su modelo de negocio lo requiera.

- En Usuario y grupo del sistema, seleccione Usar grupo predeterminado.
- Para la creación de contenedores de funciones de Lambda, elija Usar grupo por defecto.
- En Límite de memoria, escriba **`96 MB`**.
- En Tiempo de espera, escriba **`10 seconds`**.
- En Ancladas, elija Verdadero

  Para obtener más información, consulte the section called “Configuración del ciclo de vida”.

7. En Variables de entorno, cree un par clave-valor. Un par clave-valor es necesario para las funciones que interactúan con modelos MXNet en un Raspberry Pi.

   Como clave, utilice MXNET_ENGINE_TYPE. Como valor, utilice NaiveEngine.

> **Note**
>
> En sus propias funciones de Lambda definidas por el usuario, si lo desea, puede establecer la variable de entorno en el código de la función.

8. Mantenga los valores predeterminados para todas las demás propiedades y elija Agregar función de Lambda.

Paso 6: Agregar recursos al grupo de Greengrass

En este paso, va a crear recursos para el módulo de cámara y el modelo de inferencia de machine learning, y a asociar los recursos con la función de Lambda. De este modo, la función de Lambda podrá acceder a los recursos del dispositivo del núcleo.

> **Note**
>
> Si lo ejecuta en modo no contenerizado, AWS IoT Greengrass podrá acceder a la GPU y a la cámara del dispositivo sin necesidad de configurar estos recursos del dispositivo.

En primer lugar, cree dos recursos de dispositivos locales para la cámara: uno para la memoria compartida y otro para la interfaz del dispositivo. Para obtener más información sobre el acceso a los recursos locales, consulte Acceder a recursos locales con conectores y funciones de Lambda.

1. En la página de configuración del grupo, elija la pestaña Recursos.
2. En la pestaña Recursos locales, elija Añadir recurso local.
3. En la página Añadir un recurso local, utilice los siguientes valores:

   - En Nombre del recurso, escriba **`videoCoreSharedMemory`**.
   - En Tipo de recurso, elija Dispositivo.
   - En Ruta del dispositivo, escriba **`/dev/vcsm`**.

     La ruta del dispositivo es la ruta local completa del recurso del dispositivo. Esta ruta solo puede hacer referencia a un dispositivo de carácter o un dispositivo de bloques situado bajo `/dev`.

- En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.

  La opción Permisos de acceso a los archivos del propietario del grupo le permite conceder al proceso de Lambda permisos de acceso a archivos adicionales. Para obtener más información, consulte Group owner file access permission (Permiso de acceso a los archivos del propietario del grupo).

4. A continuación, añada un recurso de dispositivo local para la interfaz de la cámara.
5. Seleccione Añadir recurso local.
6. En la página Añadir un recurso local, utilice los siguientes valores:

   - En Nombre del recurso, escriba **`videoCoreInterface`**.
   - En Tipo de recurso, elija Dispositivo.
   - En Ruta del dispositivo, escriba **`/dev/vchiq`**.
   - En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.
7. En la parte inferior de la página, elija Añadir otro recurso.

Ahora, añada el modelo de inferencia como recurso de machine learning. Este paso incluye cargar el paquete de modelo `squeezenet.zip` en Amazon S3.

1. En la pestaña Recursos de su grupo, en la sección Machine learning, seleccione Añadir recurso de machine learning.
2. En la página Añadir un recurso de machine learning, en Nombre del recurso, escriba **`squeezenet_model`**.
3. En Origen del modelo, elija Usar un modelo almacenado en S3, como un modelo optimizado mediante el Compilador de Aprendizaje Profundo.
4. Para el URI de S3, introduzca una ruta en la que se guarde el bucket de S3.
5. Elija Browse S3 (Examinar S3). Esto abre una nueva pestaña en la consola de Amazon S3.

6. En la pestaña de la consola de Amazon S3 cargue su archivo `squeezenet.zip` en un bucket de Amazon S3. Para obtener información, consulte ¿Cómo puedo cargar archivos y carpetas en un bucket de S3? en la Guía del usuario de Amazon Simple Storage Service.

   > **Note**
   >
   > Para que se pueda acceder al bucket de S3, el nombre del bucket debe contener la cadena **`greengrass`** y el bucket debe estar en la misma región que usted utiliza AWS IoT Greengrass. Elija un nombre único (como **`greengrass-bucket-`*`user-id`*`-`*`epoch-time`***). No utilice un punto (`.`) en el nombre del bucket.

7. En la pestaña de la consola de AWS IoT Greengrass, localice y elija su bucket de S3. Localice y cargue el archivo `squeezenet.zip` y elija Select (Seleccionar). Es posible que tenga que elegir Refresh (Actualizar) para actualizar la lista de buckets y archivos disponibles.
8. En Destination path (Ruta de destino), escriba **`/greengrass-machine-learning/mxnet/squeezenet`**.

   Este es el destino para el modelo local en el espacio de nombres del tiempo de ejecución de Lambda. Cuando se implementa el grupo, AWS IoT Greengrass recupera el paquete del modelo de origen y, a continuación, extrae el contenido en el directorio especificado. La función de Lambda de ejemplo de este tutorial ya está configurada para utilizar esta ruta (en la variable `model_path`).
9. En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Sin grupo del sistema.
10. Seleccione Add resource (Añadir recurso).

Uso de modelos entrenados por SageMaker

Este tutorial utiliza un modelo almacenado en Amazon S3, pero también puedes utilizar fácilmente modelos de SageMaker. La consola AWS IoT Greengrass tiene integrada la integración con SageMaker, por lo que no necesitas subir manualmente estos modelos a Amazon S3. Para conocer los requisitos y las restricciones del uso de modelos de SageMaker, consulte the section called “Orígenes de modelos admitidos”.

Para usar un modelo de SageMaker:

- En Origen del modelo, seleccione Utilizar un modelo entrenado en SageMakerAWS y, a continuación, elija el nombre del trabajo de entrenamiento del modelo.
- Para ruta de destino, introduzca la ruta al directorio donde su función de Lambda busca el modelo.

Paso 7: Agregar una suscripción al grupo de Greengrass

En este paso, va a agregar una suscripción al grupo. Esta suscripción permite a la función de Lambda enviar los resultados de predicción a AWS IoT mediante la publicación en un tema MQTT.

1. En la página de configuración del grupo, elija la pestaña Suscripciones y, a continuación, elija Añadir suscripción.
2. En la página Detalles de la suscripción, configure el origen y el destino de la siguiente manera:
   a. En Tipo de origen, elija Función de lambda y, a continuación, greengrassObjectClassification.
   b. En Tipo de destino, elija Servicio y, a continuación, Nube de IoT.
3. En Filtro por temas, introduzca **`hello/world`** y, a continuación, seleccione Crear suscripción.

Paso 8: Implementar el grupo de Greengrass

En este paso, va a implementar la versión actual de la definición del grupo en el dispositivo del núcleo de Greengrass. La definición contiene la función de Lambda, los recursos y las configuraciones de suscripciones que ha añadido.

1. Asegúrese de que el núcleo de AWS IoT Greengrass se está ejecutando. Ejecute los siguientes comandos en el terminal de Raspberry Pi según sea necesario.
   a. Para comprobar si el daemon está en ejecución:

      ```
      ps aux | grep -E 'greengrass.*daemon'
      ```

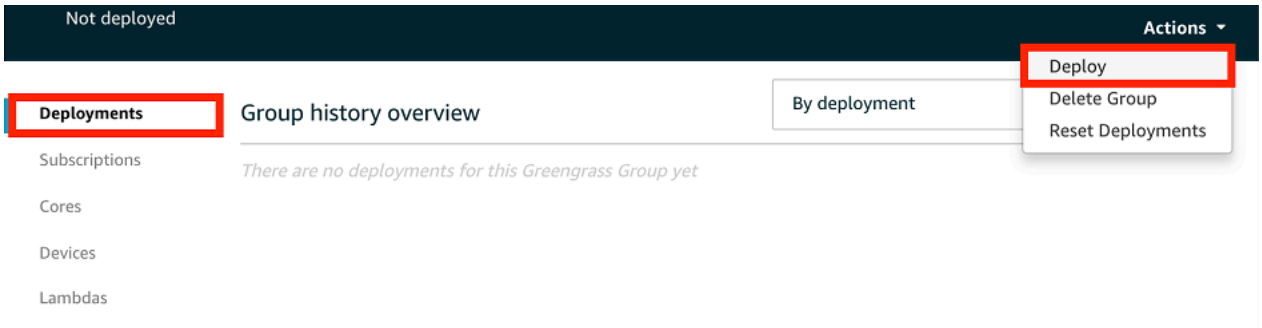
      Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, el daemon está en ejecución.

> **Note**
>
> La versión que figura en la ruta depende de la versión del software AWS IoT Greengrass Core que esté instalada en el dispositivo del núcleo.

b. Iniciar el daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. En la página de configuración de grupo, elija Implementar.



3. En la pestaña Funciones de Lambda, en la sección Funciones de Lambda del sistema, seleccione Detector IP y elija Editar.
4. En el cuadro de diálogo Editar configuración del detector IP, seleccione Detectar y anular automáticamente los puntos de conexión del agente MQTT.
5. Seleccione Save.

Esto permite a los dispositivos adquirir automáticamente la información de conexión del dispositivo principal, como la dirección IP, el DNS y el número de puerto. Se recomienda la detección automática, pero AWS IoT Greengrass también es compatible con puntos de conexión especificados manualmente. Solo se le solicitará el método de detección la primera vez que se implemente el grupo.

> **Note**
>
> Si se le solicita, conceda permiso para crear el rol de servicio de Greengrass y asócielo a su Cuenta de AWS en el Región de AWS actual. Este rol permite a AWS IoT Greengrass acceder a los servicios de AWS.

En la página Deployments (Implementaciones), se muestra la marca temporal, el ID de versión y el estado de la implementación. Una vez terminada, la implementación debería mostrar el estado Completado.

Para obtener más información sobre las implementaciones, consulte Implementación de grupos de AWS IoT Greengrass. Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

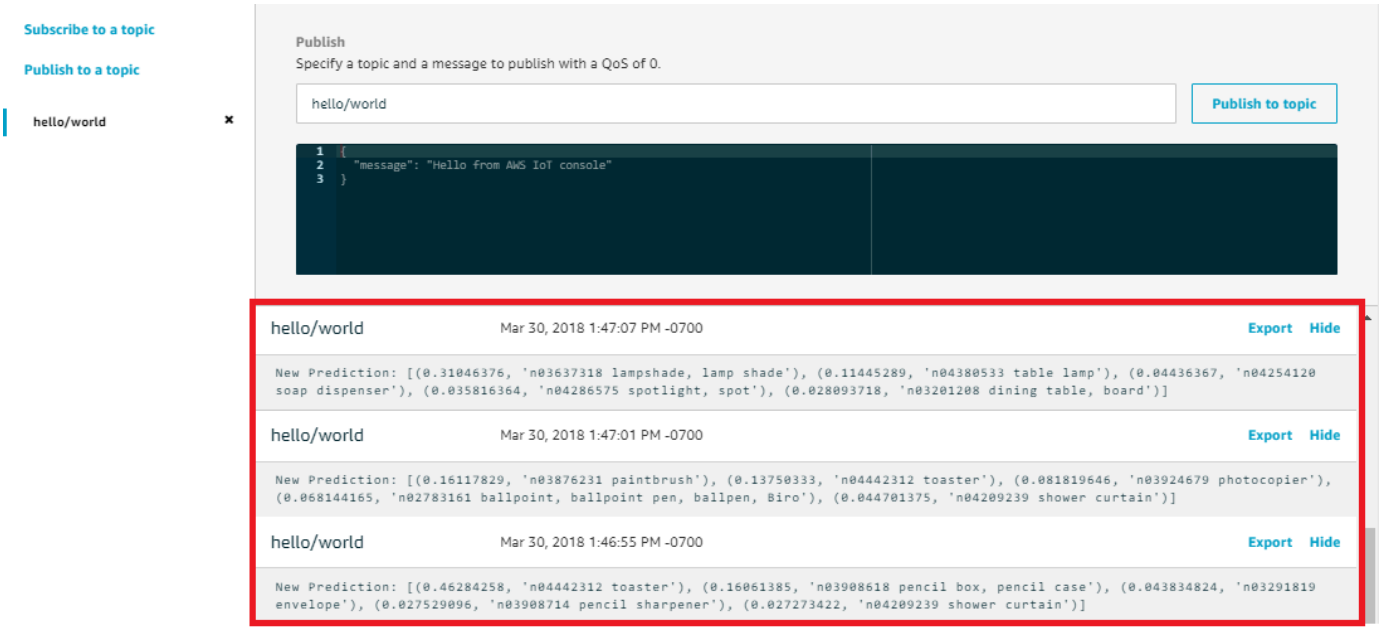
Paso 9: Probar la aplicación de inferencias

Ahora puede verificar si la implementación se ha configurado correctamente. Para ello, suscríbase al tema `hello/world` y vea los resultados de la predicción que publica la función de Lambda.

> **Note**
>
> Si se conecta un monitor al Raspberry Pi, la imagen en directo de la cámara se muestra en una ventana de vista previa.

1. En la consola AWS IoT, en Prueba, seleccione Cliente de prueba MQTT.
2. En Subscriptions (Suscripciones), utilice los siguientes valores:
   - Como tema de la suscripción, utilice hello/world.
   - En Configuración adicional, para la visualización de la carga útil MQTT, seleccione Visualizar las cargas útiles como cadenas.
3. Elija Subscribe.

   Si la prueba se realiza correctamente, los mensajes de la función de Lambda aparecen en la parte inferior de la página. Cada mensaje contiene los cinco primeros resultados de predicción de la imagen, con este formato: probabilidad, ID de la clase predicha y nombre de la clase correspondiente.



Solución de problemas relacionados con la inferencia de machine learning de AWS IoT Greengrass

Si la prueba no se realiza correctamente, puede seguir estos pasos de solución de problemas. Ejecute los comandos en el terminal de Raspberry Pi.

Comprobación de los registros de error

1. Cambie al usuario raíz y vaya al directorio `log`. El acceso a los registros de AWS IoT Greengrass requiere permisos raíz.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. En el directorio de `system`, marque `runtime.log` o `python_runtime.log`.

   En el directorio de `user/`*`region`*`/`*`account-id`*, marque `greengrassObjectClassification.log`.

   Para obtener más información, consulte the section called “Solución de problemas con los registros”.

Error de "desempaquetado" en runtime.log

Si `runtime.log` contiene un error similar al siguiente, asegúrese de que el archivo `tar.gz` del paquete del modelo de origen tiene un directorio principal.

```
Greengrass deployment error: unable to download the artifact model-arn: Error while
 processing.
Error while unpacking the file from /tmp/greengrass/artifacts/model-arn/path to /
greengrass/ggc/deployment/path/model-arn,
error: open /greengrass/ggc/deployment/path/model-arn/squeezenet/
squeezenet_v1.1-0000.params: no such file or directory
```

Si el paquete no tiene un directorio principal con los archivos del modelo, utilice el comando siguiente para volver a comprimir el modelo:

```
tar -zcvf model.tar.gz ./model
```

Por ejemplo:

```
#$ tar -zcvf test.tar.gz ./test
./test
./test/some.file
./test/some.file2
./test/some.file3
```

> **Note**
>
> No incluya caracteres `/*` en este comando.

Compruebe que la función de Lambda se ha implementado correctamente

1. Muestre el contenido de la Lambda implementada en el directorio `/lambda`. Reemplace los valores de los marcadores de posición antes de ejecutar el comando .

   ```
   cd /greengrass/ggc/deployment/lambda/
   arn:aws:lambda:region:account:function:function-name:function-version
   ```

```
ls -la
```

2. Compruebe que el directorio contiene el mismo contenido que el paquete de implementación `greengrassObjectClassification.zip` que cargó en el Paso 4: Crear y publicar una función de Lambda.

   Asegúrese también de que los archivos `.py` y las dependencias se encuentran en la raíz del directorio.

Compruebe que el modelo de inferencia se ha implementado correctamente

1. Busque el número de identificación de proceso (PID) del proceso de tiempo de ejecución de Lambda:

   ```
   ps aux | grep 'lambda-function-name*'
   ```

   En el resultado, el PID aparece en la segunda columna de la línea correspondiente al proceso de tiempo de ejecución de Lambda.

2. Entre en el espacio de nombres del tiempo de ejecución de Lambda. Asegúrese de que sustituye el valor del marcador de posición *`pid`* antes de ejecutar el comando.

   > **Note**
   >
   > Este directorio y su contenido se encuentran en el espacio de nombres del tiempo de ejecución de Lambda, por lo que no son visibles en un espacio de nombres de Linux normal.

   ```
   sudo nsenter -t pid -m /bin/bash
   ```

3. Muestre el contenido del directorio local que ha especificado para el recurso de machine learning.

   ```
   cd /greengrass-machine-learning/mxnet/squeezenet/
   ls -ls
   ```

Debería ver los siguientes archivos:

```
32 -rw-r--r-- 1 ggc_user ggc_group   31675 Nov 18 15:19 synset.txt
32 -rw-r--r-- 1 ggc_user ggc_group   28707 Nov 18 15:19 squeezenet_v1.1-symbol.json
4832 -rw-r--r-- 1 ggc_user ggc_group 4945062 Nov 18 15:19
 squeezenet_v1.1-0000.params
```

Pasos siguientes

A continuación, explore otras aplicaciones de inferencias. AWS IoT Greengrass proporciona otras funciones de Lambda que puede utilizar para probar la inferencia local. Puede encontrar el paquete con los ejemplos en la carpeta de bibliotecas precompiladas que descargó en el the section called “Instalar el marco MXNet”.

Configuración de Intel Atom

Para ejecutar este tutorial en un dispositivo Intel Atom, debe proporcionar imágenes de origen, configurar la función de Lambda y agregar otro recurso del dispositivo local. Para utilizar la GPU como inferencia, compruebe que el siguiente software está instalado en el dispositivo:

- OpenCL versión 1.0 o posterior
- Python 3.7 y pip

> **Note**
>
> Si el dispositivo se ha precompilado con Python 3.6, puede crear un symlink a Python 3.7, en su lugar. Para obtener más información, consulte Step 2.

- NumPy
- OpenCV en Wheels

1. Descargue imágenes PNG o JPG estáticas para la función de Lambda que se va a utilizar para la clasificación de imágenes. El ejemplo funciona mejor con archivos de imágenes pequeños.

   Guarde los archivos de imágenes en el directorio que contiene el archivo `greengrassObjectClassification.py` (o en un subdirectorio de este directorio). Se

encuentra en el paquete de implementación de la función de Lambda que cargó en the section called “Crear y publicar una función de Lambda.”.

> **Note**
>
> Si está utilizando AWS DeepLens, puede emplear la cámara integrada o montar su propia cámara para realizar la inferencia de imágenes capturadas en lugar de imágenes estáticas. Sin embargo, le recomendamos encarecidamente que empiece primero con imágenes estáticas.
> Si utiliza una cámara, asegúrese de que el paquete `awscam` de APT esté instalado y actualizado. Para obtener más información, consulte Actualización del dispositivo AWS DeepLens en la Guía para desarrolladores de AWS DeepLens.

2. Si no está utilizando Python 3.7, asegúrese de crear un enlace simbólico de Python 3.x a Python 3.7. Esto configurará el dispositivo para que utilice Python 3 con AWS IoT Greengrass. Ejecute el siguiente comando para localizar la instalación de Python:

```
which python3
```

Ejecute el siguiente comando para crear el symlink:

```
sudo ln -s path-to-python-3.x/python3.x path-to-python-3.7/python3.7
```

Reinicie el dispositivo.

3. Edite la configuración de la función de Lambda. Siga el procedimiento indicado en the section called “Añadir la función de Lambda al grupo”.

> **Note**
>
> Le recomendamos que ejecute la función de Lambda sin creación de contenedores, a menos que su modelo de negocio lo requiera. Esto permite el acceso a la GPU y la cámara del dispositivo sin necesidad de configurar los recursos del dispositivo.
> Si ejecuta sin creación de contenedores, también debe conceder acceso raíz a las funciones de Lambda AWS IoT Greengrass.

   a. Para ejecutar sin creación de contenedores:

- En Usuario y grupo del sistema, elija **`Another user ID/group ID`**. En ID de usuario del sistema, introduzca **`0`**. Para el ID de grupo del sistema, introduzca **`0`**.

  Esto permite que la función de Lambda se ejecute como raíz. Para obtener más información sobre cómo ejecutar un trabajo, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

  > **Tip**
  >
  > También debe actualizar el archivo `config.json` para conceder acceso raíz a la función de Lambda. Para informarse sobre este procedimiento, consulte the section called “Ejecución de una función de Lambda como raíz”.

- Para la creación de contenedores de la función de Lambda, elija Sin contenedor.

  Para obtener más información sobre la ejecución sin creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.

- Actualice el valor de Tiempo de espera a 5 segundos. De este modo, se asegurará de que la solicitud no supere el tiempo de inactividad demasiado pronto. La configuración tarda unos minutos en ejecutar la inferencia.
- En Ancladas, elija Verdadero.
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.
- Para Lambda lifecycle (Ciclo de vida de Lambda), elija Make this function long-lived and keep it running indefinitely (Prolongar la vida útil de esta función y mantenerla en ejecución indefinidamente).

b. Para ejecutarlo en modo contenerizado, en su lugar:

> **Note**
>
> No recomendamos ejecutarlo en modo contenerizado a menos que su modelo de negocio lo requiera.

- Actualice el valor de Tiempo de espera a 5 segundos. De este modo, se asegurará de que la solicitud no supere el tiempo de inactividad demasiado pronto. La configuración tarda unos minutos en ejecutar la inferencia.
- En Ancladas, elija Verdadero
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.

4. Si se ejecuta en modo contenerizado, añada el recurso de dispositivo local necesario para conceder acceso a su GPU de dispositivo.

> **Note**
>
> Si lo ejecuta en modo no contenerizado, AWS IoT Greengrass podrá acceder a la GPU del dispositivo sin necesidad de configurar este recurso del dispositivo.

   a. En la página de configuración del grupo, elija la pestaña Recursos.
   b. Seleccione Añadir recurso local.
   c. Defina el recurso:

      - En Nombre del recurso, escriba **`renderD128`**.
      - En Tipo de recurso, elija Dispositivo local.
      - En Device path (Ruta del dispositivo), escriba **`/dev/dri/renderD128`**.
      - En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.
      - En Afiliaciones de funciones de Lambda, conceda Acceso de lectura y escritura a la función de Lambda.

Configuración de un dispositivo NVIDIA Jetson TX2

Para ejecutar este tutorial en un dispositivo NVIDIA Jetson TX2, proporcione las imágenes de origen y configure la función de Lambda. Si está utilizando la GPU, también debe agregar recursos de dispositivos locales.

1. Compruebe que el dispositivo Jetson está configurado de forma que permita instalar el software AWS IoT Greengrass Core. Para obtener más información sobre la configuración del proyecto, consulte the section called “Configuración de otros dispositivos”.
2. Abra la documentación de MXNet, vaya a Installing MXNet on a Jetson y siga las instrucciones para instalar MXNet en el dispositivo Jetson.

   > **Note**
   >
   > Si desea compilar MxNet desde el origen, siga las instrucciones para crear la biblioteca compartida. Modifique los siguientes ajustes en el archivo `config.mk` para trabajar con un dispositivo Jetson TX2:
   >
   > - Agréguela `-gencode arch=compute-62, code=sm_62` a la opción `CUDA_ARCH`.
   > - Active CUDA.
   >
   > ```
   > USE_CUDA = 1
   > ```

3. Descargue imágenes PNG o JPG estáticas para la función de Lambda que se va a utilizar para la clasificación de imágenes. La aplicación funciona mejor con archivos de imágenes pequeños. Si lo prefiere, puede instrumentar una cámara en la placa Jetson para capturar las imágenes de origen.

   Guarde los archivos de la imagen en la carpeta que contiene el archivo `greengrassObjectClassification.py`. También puede guardarlos en un subdirectorio de esta carpeta. Este directorio esté en el paquete de implementación de funciones de Lambda que ha cargado en the section called “Crear y publicar una función de Lambda.”.
4. Cree un symlink entre Python 3.7 y Python 3.6 para poder usar Python 3 con AWS IoT Greengrass. Ejecute el siguiente comando para localizar la instalación de Python:

   ```
   which python3
   ```

   Ejecute el siguiente comando para crear el symlink:

   ```
   sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
   ```

   Reinicie el dispositivo.
5. Asegúrese de que la cuenta `ggc_user` del sistema puede usar el marco MxNet:

```
“sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

6. Edite la configuración de la función de Lambda. Siga el procedimiento indicado en the section called “Añadir la función de Lambda al grupo”.

> **Note**
>
> Le recomendamos que ejecute la función de Lambda sin creación de contenedores, a menos que su modelo de negocio lo requiera. Esto permite el acceso a la GPU y la cámara del dispositivo sin necesidad de configurar los recursos del dispositivo. Si ejecuta sin creación de contenedores, también debe conceder acceso raíz a las funciones de Lambda AWS IoT Greengrass.

a. Para ejecutar sin creación de contenedores:

- En Usuario y grupo del sistema, elija **`Another user ID/group ID`**. En ID de usuario del sistema, introduzca **`0`**. Para el ID de grupo del sistema, introduzca **`0`**.

  Esto permite que la función de Lambda se ejecute como raíz. Para obtener más información sobre cómo ejecutar un trabajo, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

  > **Tip**
  >
  > También debe actualizar el archivo `config.json` para conceder acceso raíz a la función de Lambda. Para informarse sobre este procedimiento, consulte the section called “Ejecución de una función de Lambda como raíz”.

- Para la creación de contenedores de la función de Lambda, elija Sin contenedor.

  Para obtener más información sobre la ejecución sin creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.

- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.
- En Variables de entorno, añada los siguientes pares clave-valor a su función de Lambda. Esto configurará AWS IoT Greengrass de modo que permita usar el marco MxNet.

Clave	Valor
PATH	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

b. Para ejecutarlo en modo contenerizado, en su lugar:

> **Note**
>
> No recomendamos ejecutarlo en modo contenerizado a menos que su modelo de negocio lo requiera.

- Aumente el valor de Memory limit (Límite de memoria). Utilice 500 MB para la CPU o, como mínimo, 2000 MB para la GPU.
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.
- En Variables de entorno, añada los siguientes pares clave-valor a su función de Lambda. Esto configurará AWS IoT Greengrass de modo que permita usar el marco MxNet.

Clave	Valor
PATH	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH
CUDA_HOME	/usr/local/cuda

Clave	Valor
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

7. Si se ejecuta en modo contenerizado, añada los recursos de dispositivo local siguientes para conceder acceso a su GPU de dispositivo. Siga el procedimiento indicado en the section called “Agregar recursos al grupo”.

   > **Note**
   >
   > Si lo ejecuta en modo no contenerizado, AWS IoT Greengrass podrá acceder a la GPU del dispositivo sin necesidad de configurar este recurso del dispositivo.

   Para cada recurso:

   - En Tipo de recurso, elija Dispositivo.
   - En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.

Nombre	Ruta del dispositivo
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic

Nombre	Ruta del dispositivo
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. Si se ejecuta en modo contenerizado, añada el siguiente recurso de volumen local para conceder acceso a la cámara de su dispositivo. Siga el procedimiento indicado en the section called “Agregar recursos al grupo”.

   > **Note**
   >
   > Si lo ejecuta en modo no contenerizado, AWS IoT Greengrass podrá acceder a la cámara del dispositivo sin necesidad de configurar este recurso del volumen.

   - En Resource type (Tipo de recurso), elija Device (Dispositivo).
   - En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.

Nombre	Ruta de origen	Destination path
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

Cómo configurar la inferencia de machine learning optimizado mediante AWS Management Console

Para seguir los pasos de este tutorial, debe utilizar Núcleo de AWS IoT Greengrass versión 1.10 o posterior.

Puede utilizar el compilador de aprendizaje profundo SageMaker Neo para optimizar la eficacia de predicción de los modelos nativos de inferencia de machine learning en los marcos Tensorflow, Apache MXNet, PyTorch, ONNX y XGBoost para obtener una huella más pequeña y un rendimiento más rápido. A continuación, puede descargar el modelo optimizado e instalar el tiempo de ejecución

del aprendizaje profundo SageMaker Neo y desplegarlos en sus dispositivos AWS IoT Greengrass para una inferencia más rápida.

En este tutorial se describe cómo utilizar la AWS Management Console para configurar un grupo de Greengrass para ejecutar un ejemplo de inferencias de Lambda que reconoce imágenes de una cámara localmente, sin enviar datos a la nube. El ejemplo de inferencia obtiene acceso al módulo de cámara en un Raspberry Pi. En este tutorial, se descarga un modelo preempaquetado entrenado por Resnet-50 y optimizado en el compilador de aprendizaje profundo Neo. A continuación, utilice el modelo para realizar la clasificación de imagen local en el dispositivo de AWS IoT Greengrass.

El tutorial contiene los siguientes pasos generales:

1. Configuración de Raspberry Pi
2. Instalar el entorno de ejecución de aprendizaje profundo de Neo
3. Crear una función de Lambda de inferencia.
4. Añadir la función de Lambda al grupo
5. Agregar al grupo un recurso del modelo optimizado para Neo
6. Agregar un recurso de dispositivo de cámara al grupo
7. Agregar suscripciones al grupo
8. Implementar el grupo
9. Prueba del ejemplo

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Raspberry Pi 4 modelo B o Raspberry Pi 3 modelo B/B+, configurados y configurados para su uso con AWS IoT Greengrass. Para configurar su Raspberry Pi con AWS IoT Greengrass, ejecute el script de configuración del dispositivo Greengrass o asegúrese de haber completado el módulo 1 y el módulo 2 de Empezar con AWS IoT Greengrass.

  > **Note**
  >
  > Es posible que la Raspberry Pi requiera una fuente de alimentación de 2,5 A para ejecutar los marcos de aprendizaje profundo que se utilizan normalmente para la clasificación de

imágenes. Una fuente de alimentación con una potencia inferior podría provocar el reinicio del dispositivo.

- Módulo de cámara Raspberry Pi V2 de 8 megapíxeles, 1080p. Para obtener información sobre cómo configurar la cámara, consulte Connecting the camera en la documentación de Raspberry Pi.
- Un grupo de Greengrass y un núcleo de Greengrass. Para obtener información sobre cómo crear un grupo o dispositivo del núcleo de Greengrass, consulte Empezar con AWS IoT Greengrass.

> **Note**
>
> En este tutorial se utiliza un dispositivo Raspberry Pi, pero AWS IoT Greengrass es compatible con otras plataformas, como Intel Atom y NVIDIA Jetson TX2. Si utiliza el ejemplo de Intel Atom, es posible que necesite instalar Python 3.6 en lugar de Python 3.7. Si necesita más información acerca de cómo configurar el dispositivo para poder instalar el software AWS IoT Greengrass Core, consulte the section called “Configuración de otros dispositivos”. Para las plataformas de terceros que AWS IoT Greengrass no son compatibles, debe ejecutar la función de Lambda en modo no contenerizado. Para ejecutarla en modo no contenerizado, debe ejecutar la función de Lambda como raíz. Para obtener más información, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.” y the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

Paso 1: Configurar el Raspberry Pi

En este paso, va a instalar actualizaciones del sistema operativo Raspbian, el software del módulo de cámara y las dependencias de Python, y va a habilitar la interfaz de la cámara.

Ejecute los siguientes comandos en el terminal de Raspberry Pi.

1. Instale las actualizaciones en Raspbian.

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

2. Instale la interfaz `picamera` del módulo de cámara y las demás bibliotecas de Python que sean necesarias para este tutorial.

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

Valide la instalación:

- Asegúrese de que la instalación de Python 3.7 incluye pip.

  ```
  python3 -m pip
  ```

  Si pip no está instalado, descárguelo del sitio web de pip y ejecute el siguiente comando.

  ```
  python3 get-pip.py
  ```

- Asegúrese de que la versión de Python es 3.7 o superior.

  ```
  python3 --version
  ```

  Si en la salida aparece una versión anterior, ejecute el siguiente comando.

  ```
  sudo apt-get install -y python3.7-dev
  ```

- Asegúrese de que Setuptools y Picamera se han instalado correctamente.

  ```
  sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
  sudo -u ggc_user bash -c 'python3 -c "import picamera"'
  ```

  Si el resultado no contiene errores, la validación es correcta.

> **Note**
>
> Si el ejecutable de Python instalado en el dispositivo es `python3.7`, utilice `python3.7` en lugar de `python3` con los comandos de este tutorial. Asegúrese de que la instalación de pip corresponde a la versión `python3` o `python3.7` correcta para evitar errores de dependencia.

3. Reinicie el Raspberry Pi.

```
sudo reboot
```

4. Abra la herramienta de configuración de Raspberry Pi.

```
sudo raspi-config
```

5. Utilice las teclas de flecha para abrir Interfacing Options (Opciones de interfaz) y habilitar la interfaz de la cámara. Si se le solicita, permita que el dispositivo se reinicie.
6. Utilice el siguiente comando para probar la configuración de la cámara.

```
raspistill -v -o test.jpg
```

Se abre una ventana de vista previa en el Raspberry Pi, se guarda una imagen denominada `test.jpg` en el directorio actual y se muestra información sobre la cámara en el terminal de Raspberry Pi.

Paso 2: Instalar el entorno de ejecución de aprendizaje profundo de Amazon SageMaker Neo

En este paso, instale el tiempo de ejecución del aprendizaje profundo Neo (DLR) en su Raspberry Pi.

> **Note**
>
> Para este tutorial, recomendamos instalar la versión 1.1.0.

1. Inicie sesión en su Raspberry Pi de forma remota.

```
ssh pi@your-device-ip-address
```

2. Abra la documentación del DLR, abra Installing DLR y busque la URL del wheel para los dispositivos Raspberry Pi. A continuación, siga las instrucciones para instalar DLR en su dispositivo. Por ejemplo, puede utilizar pip:

```
pip3 install rasp3b-wheel-url
```

3. Después de instalar el DLR, valide la siguiente configuración:

   - Asegúrese de que la cuenta `ggc_user` del sistema puede usar la biblioteca del DLR.

```
sudo -u ggc_user bash -c 'python3 -c "import dlr"'
```

- Compruebe que NumPy esté instalado.

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

Paso 3: Crear una función de Lambda de inferencia

En este paso, va a crear un paquete de implementación de funciones de Lambda y una función de Lambda. A continuación, publicará una versión de la función y creará un alias.

1. En su equipo, descargue el ejemplo del DLR para Raspberry Pi de the section called “Ejemplos de aprendizaje automático”.
2. Descomprima el archivo `dlr-py3-armv7l.tar.gz` descargado.

   ```
   cd path-to-downloaded-sample
   tar -xvzf dlr-py3-armv7l.tar.gz
   ```

   El directorio `examples` del paquete de ejemplo extraído contiene el código de la función y las dependencias.

   - `inference.py` es el código de inferencia que se utiliza en este tutorial. Puede usar este código como plantilla para crear su propia función de inferencia.
   - `greengrasssdk` es la versión 1.5.0 del AWS IoT Greengrass Core SDK para Python.

   > **Note**
   >
   > Si hay una nueva versión disponible, puede descargarla y actualizar la versión del SDK del paquete de implementación. Para obtener más información, AWS IoT Greengrassvea SDK de Python en GitHub.

3. Comprima el contenido del directorio `examples` en un archivo llamado `optimizedImageClassification.zip`. Este es el paquete de implementación.

   ```
   cd path-to-downloaded-sample/dlr-py3-armv7l/examples
   zip -r optimizedImageClassification.zip .
   ```

El paquete de implementación contiene el código de la característica y las dependencias. Esto incluye el código que invoca las API de Python del tiempo de ejecución del aprendizaje profundo de Neo para realizar inferencias con los modelos de compilador de aprendizaje profundo de Neo.

> **Note**
>
> Asegúrese de que los archivos `.py` y las dependencias se encuentran en la raíz del directorio.

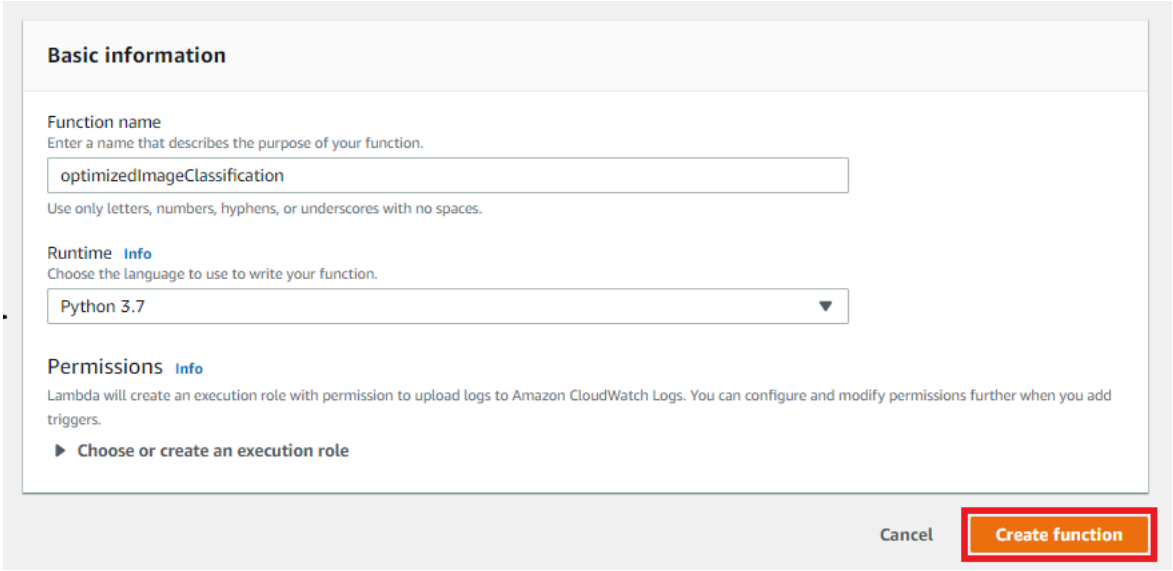
4. Ahora, añada función de Lambda al grupo de Greengrass.

   Abra Funciones de la página de consolas de Lambda y elija Crear función.

5. Elija Autor desde cero y utilice los siguientes valores para crear su función:

   - En Function name (Nombre de la característica), introduzca **`optimizedImageClassification`**.
   - En Runtime (Tiempo de ejecución), elija Python 3.7.

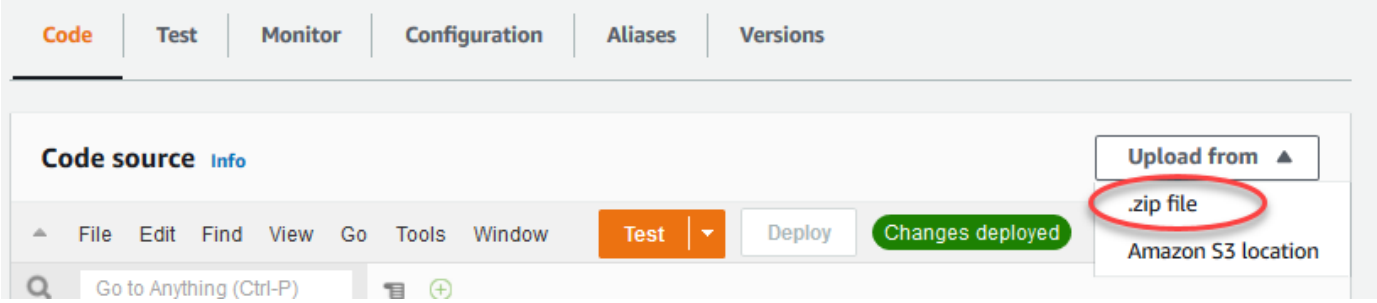
   En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol.



6. Elija Crear función.

Ahora, cargue el paquete de implementación de la función de Lambda y registre el controlador.

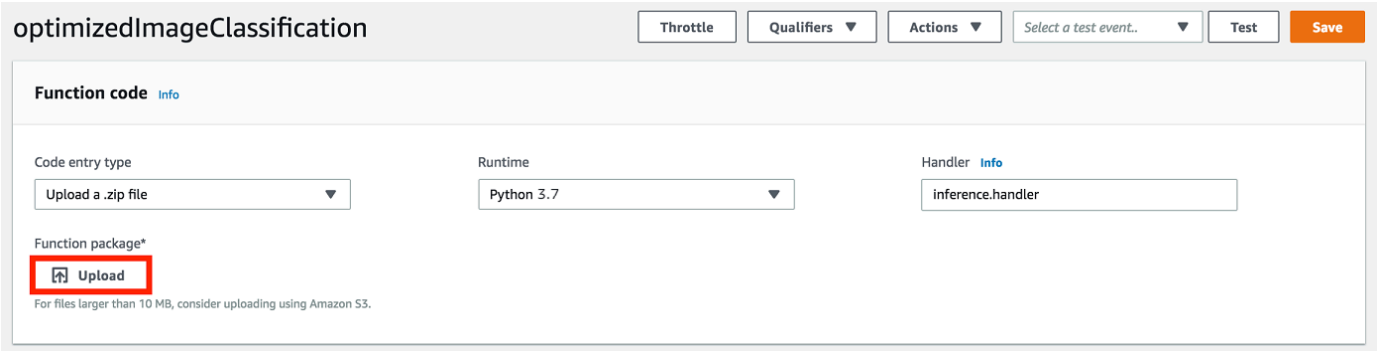
1. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip..



2. Elija su paquete de implementación `optimizedImageClassification.zip` y, a continuación, seleccione Guardar.
3. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

   - En Runtime (Tiempo de ejecución), elija Python 3.7.
   - En Handler (Controlador), escriba **`inference.handler`**.

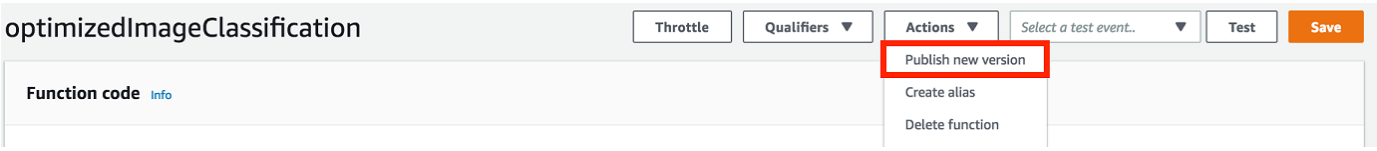
   Seleccione Save.



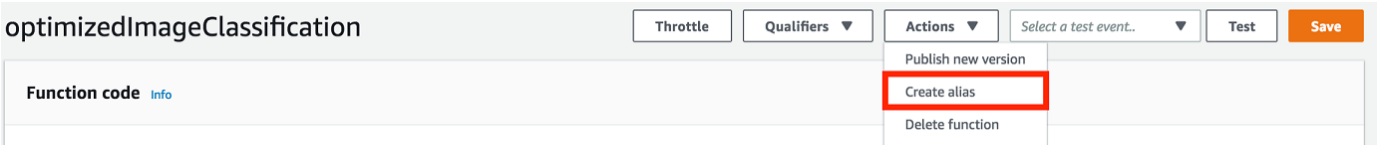
A continuación, publique la primera versión de la función de Lambda. A continuación, cree un alias para la versión.

> **Note**
>
> Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

1. En el menú Actions, elija Publish new version.



2. En Version description (Descripción de versión), escriba **`First version`** y, a continuación, elija Publish (Publicar).
3. En la página de configuración de optimizedImageClassification: 1, en el menú Actions (Acciones), elija Create alias (Crear alias).



4. En la página Create a new alias, utilice los valores siguientes:
   - En Name (Nombre), ingrese **`mlTestOpt`**.
   - En Version (Versión), escriba **`1`**.

   > **Note**
   >
   > AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST.

5. Seleccione Create (Crear).

Ahora, añada función de Lambda al grupo de Greengrass.

Paso 4: Añadir la función de Lambda al grupo de Greengrass

En este paso, va a agregar la función de Lambda al grupo y a configurar su ciclo de vida.

Primero, añada función de Lambda al grupo de Greengrass.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. En la página de configuración de grupos, elija la pestaña Funciones de Lambda y seleccione Añadir.
3. Elija la Función de Lambda y seleccione OptimizediMageClassification.
4. En la versión de la función de Lambda, elija el alias de la versión que publicó.

A continuación, configure el ciclo de vida de la función de Lambda.

1. En la sección de configuración de la función de Lambda, realice las siguientes actualizaciones.

   > **Note**
   >
   > Le recomendamos que ejecute la función de Lambda sin creación de contenedores, a menos que su modelo de negocio lo requiera. Esto permite el acceso a la GPU y la cámara del dispositivo sin necesidad de configurar los recursos del dispositivo. Si ejecuta sin creación de contenedores, también debe conceder acceso raíz a las funciones de LambdaAWS IoT Greengrass.

   a. Para ejecutar sin creación de contenedores:

      - En Usuario y grupo del sistema, elija **`Another user ID/group ID`**. En ID de usuario del sistema, introduzca **`0`**. Para el ID de grupo del sistema, introduzca **`0`**.

        Esto permite que la función de Lambda se ejecute como raíz. Para obtener más información sobre cómo ejecutar un trabajo, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

> **Tip**
>
> También debe actualizar el archivo `config.json` para conceder acceso raíz a la función de Lambda. Para informarse sobre este procedimiento, consulte the section called “Ejecución de una función de Lambda como raíz”.

- Para la creación de contenedores de funciones de Lambda, elija Sin contenedor.

  Para obtener más información sobre la ejecución sin creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.
- En Tiempo de espera, escriba **`10 seconds`**.
- En Ancladas, elija Verdadero

  Para obtener más información, consulte the section called “Configuración del ciclo de vida”.
- En Parámetro adicional, para Acceso de lectura al directorio /sys, elija Activado.

b. Para ejecutarlo en modo contenerizado, en su lugar:

> **Note**
>
> No recomendamos ejecutarlo en modo contenerizado a menos que su modelo de negocio lo requiera.

- En Usuario y grupo del sistema, seleccione Usar grupo predeterminado.
- Para la creación de contenedores de funciones de Lambda, elija Usar grupo por defecto.
- En Límite de memoria, escriba **`1024 MB`**.
- En Tiempo de espera, escriba **`10 seconds`**.
- En Ancladas, elija Verdadero

  Para obtener más información, consulte the section called “Configuración del ciclo de vida”.
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.

2. Elija Función de Lambda.

Paso 5: Agregar al grupo de SageMaker un recurso del modelo de optimizado para Neo

En este paso, va a crear un recurso para el modelo de inferencia de machine learning optimizado y a cargarlo en un bucket de Amazon S3. A continuación, va a buscar el modelo cargado de Amazon S3 en la consola de AWS IoT Greengrass y a asociar el recurso recién creado con la función de Lambda. Esto permite que la característica acceda a sus recursos en el dispositivo de nodo.

1. En el equipo, desplácese hasta el directorio `resnet50` del paquete de ejemplo en el que descomprimió the section called “Crear una función de Lambda de inferencia.”.

   > **Note**
   >
   > Si utiliza el ejemplo de NVIDIA Jetson, debe usar el directorio `resnet18` en el paquete de ejemplo en su lugar. Para obtener más información, consulte the section called “Configuración de un dispositivo NVIDIA Jetson TX2”.

   ```
   cd path-to-downloaded-sample/dlr-py3-armv7l/models/resnet50
   ```

   Este directorio contiene artefactos de modelos compilados previamente para un modelo de clasificación de imágenes entrenado con Resnet-50.

2. Comprima los archivos que están en el directorio `resnet50` en un archivo llamado `resnet50.zip`.

   ```
   zip -r resnet50.zip .
   ```

3. En la página de configuración de grupo de su grupo AWS IoT Greengrass, elija Recursos. Vaya a la sección Machine Learning y elija Añadir recurso de machine learning. En la página Crear un recurso de machine learning, en Nombre del recurso, escriba **`resnet50_model`**.

4. En Origen del modelo, elija Usar un modelo almacenado en S3, como un modelo optimizado mediante el Compilador de Aprendizaje Profundo.

5. En URI de S3, elija Browse S3.

> **Note**
>
> Actualmente, los modelos de SageMaker optimizados se almacenan automáticamente en Amazon S3. Puede encontrar su modelo optimizado en su bucket de Amazon S3 utilizando esta opción. Para obtener más información acerca de la optimización de modelos en SageMaker, consulte la documentación de SageMaker Neo.

6. Elija Upload a model (Cargar un modelo).
7. En la pestaña de la consola de Amazon S3 cargue su archivo zip en un bucket de Amazon S3. Para obtener información, consulte ¿Cómo puedo cargar archivos y carpetas en un bucket de S3? en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> El nombre del bucket debe contener la cadena **`greengrass`**. Elija un nombre único (como **`greengrass-dlr-bucket-`*`user-id`*`-`*`epoch-time`***). No utilice un punto (`.`) en el nombre del bucket.

8. En la pestaña de la consola de AWS IoT Greengrass, localice y elija su bucket de Amazon S3. Localice y cargue el archivo `resnet50.zip` y elija Select (Seleccionar). Es posible que tenga que actualizar la página para actualizar la lista de buckets y archivos disponibles.
9. En Ruta de destino, escriba **`/ml_model`**.

Este es el destino para el modelo local en el espacio de nombres del tiempo de ejecución de Lambda. Cuando se implementa el grupo, AWS IoT Greengrass recupera el paquete del modelo de origen y, a continuación, extrae el contenido en el directorio especificado.

> **Note**
>
> Le recomendamos encarecidamente que utilice la ruta exacta proporcionada por su ruta local. El uso de una ruta de destino de modelo local distinta en este paso hace que algunos comandos de la solución de problemas ofrecidos en este tutorial

sean inexactos. Si utiliza una ruta diferente, debe configurar una variable de entorno `MODEL_PATH` que utilice la ruta exacta que proporcione aquí. Para obtener información sobre las variables de entorno, consulte Variables de entorno de AWS Lambda.

10. Si se ejecuta en modo contenerizado:

    a. En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Especificar grupo del sistema y permisos.

    b. Elija Acceso de solo lectura y, a continuación, elija Agregar recurso.

Paso 6: Agregar un recurso de dispositivo de cámara al grupo de Greengrass

En este paso, va a crear un recurso para el módulo de cámara y a asociarlo con la función de Lambda. De este modo, la función de Lambda podrá acceder al recurso del dispositivo del núcleo.

> **Note**
>
> Si lo ejecuta en modo no contenerizado, AWS IoT Greengrass podrá acceder a la GPU y a la cámara del dispositivo sin necesidad de configurar este recurso del dispositivo.

1. En la página de configuración del grupo, elija la pestaña Recursos.
2. En la pestaña Recursos locales, elija Añadir recurso local.
3. En la página Añadir un recurso local, utilice los siguientes valores:

    - En Nombre del recurso, escriba **`videoCoreSharedMemory`**.
    - En Tipo de recurso, elija Dispositivo.
    - En Ruta del dispositivo, escriba **`/dev/vcsm`**.

      La ruta del dispositivo es la ruta local completa del recurso del dispositivo. Esta ruta solo puede hacer referencia a un dispositivo de carácter o un dispositivo de bloques situado bajo `/dev`.
    - En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.

La opción Group owner file access permission (Permiso de acceso a los archivos del propietario del grupo) le permite conceder al proceso de Lambda permisos de acceso a archivos adicionales. Para obtener más información, consulte Group owner file access permission (Permiso de acceso a los archivos del propietario del grupo).

4. En la parte inferior de la página, elija Añadir otro recurso.
5. En la pestaña Recursos, cree otro recurso local seleccionando Agregar y use los siguientes valores:
   - En Nombre del recurso, escriba **`videoCoreInterface`**.
   - En Tipo de recurso, elija Dispositivo.
   - En Ruta del dispositivo, escriba **`/dev/vchiq`**.
   - En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.
6. Seleccione Add resource (Añadir recurso).

Paso 7: Agregar suscripciones al grupo de Greengrass

En este paso, va a agregar suscripciones al grupo. Estas suscripciones permiten a la función de Lambda enviar los resultados de predicción a AWS IoT mediante la publicación en un tema MQTT.

1. En la página de configuración del grupo, elija la pestaña Suscripciones y, a continuación, elija Añadir suscripción.
2. En la página Crear una suscripción, configure el origen y el destino de la siguiente manera:
   a. En Seleccionar un origen, elija Funciones de Lambda y, a continuación, elija optimizedImageClassification.
   b. En Tipo de destino, elija Servicio y, a continuación, Nube de IoT.
   c. En Filtro por temas, introduzca **`/resnet-50/predictions`** y, a continuación, seleccione Crear suscripción.
3. Añadir una segunda suscripción. Seleccione la pestaña Suscripciones, elija Agregar suscripción y configure el origen y el destino de la siguiente manera:
   a. En la sección Seleccionar un origen, elija Servicios y luego elija Nube de IoT.

b. En Seleccionar un destino, elija Funciones de Lambda y, a continuación, elija optimizedImageClassification.

c. En Filtro por temas, introduzca **`/resnet-50/test`** y, a continuación, seleccione Crear suscripción.

Paso 8: Implementar el grupo de Greengrass

En este paso, va a implementar la versión actual de la definición del grupo en el dispositivo del núcleo de Greengrass. La definición contiene la función de Lambda, los recursos y las configuraciones de suscripciones que ha añadido.

1. Asegúrese de que el núcleo de AWS IoT Greengrass se está ejecutando. Ejecute los siguientes comandos en el terminal de Raspberry Pi según sea necesario.

   a. Para comprobar si el daemon está en ejecución:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/`*`latest-core-version`*`/bin/daemon`, el daemon está en ejecución.

   b. Iniciar el daemon:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. En la página de configuración de grupo, elija Implementar.
3. En la pestaña Funciones de Lambda, seleccione Detector IP y elija Editar.
4. En el cuadro de diálogo Editar configuración del detector IP, seleccione Detectar y anular automáticamente los puntos de conexión del agente MQTT y elija Guardar.

   Esto permite a los dispositivos adquirir automáticamente la información de conexión del dispositivo principal, como la dirección IP, el DNS y el número de puerto. Se recomienda la detección automática, pero AWS IoT Greengrass también es compatible con puntos de conexión especificados manualmente. Solo se le solicitará el método de detección la primera vez que se implemente el grupo.

> **Note**
>
> Si se le solicita, conceda permiso para crear el rol de servicio de Greengrass y asócielo a su Cuenta de AWS en el Región de AWS actual. Este rol permite a AWS IoT Greengrass acceder a los servicios de AWS.

En la página Deployments (Implementaciones), se muestra la marca temporal, el ID de versión y el estado de la implementación. Una vez terminada, la implementación debería mostrar el estado Completado.

Para obtener más información sobre las implementaciones, consulte Implementación de grupos de AWS IoT Greengrass. Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

Prueba del ejemplo de inferencia

Ahora puede verificar si la implementación se ha configurado correctamente. Para probarlo, suscríbase al tema `/resnet-50/predictions` y publique cualquier mensaje en el tema `/resnet-50/test`. Esto activa la función de Lambda para que tome una foto con su Raspberry Pi y realice la inferencia en la imagen que capture.

> **Note**
>
> Si utiliza el ejemplo de NVIDIA Jetson, no olvide utilizar los temas `resnet-18/predictions`y `resnet-18/test` en su lugar.

> **Note**
>
> Si se conecta un monitor al Raspberry Pi, la imagen en directo de la cámara se muestra en una ventana de vista previa.

1. En la página de inicio de la consola AWS IoT, en Probar, elija cliente de prueba MQTT.

2. Para Suscripciones, elija Suscribirse a un tema. Use los siguientes valores. No cambie los valores predeterminados de las opciones restantes.

   - Para Tema de suscripción, escriba **`/resnet-50/predictions`**.
   - En Configuración adicional, para la visualización de la carga útil MQTT, seleccione Visualizar las cargas útiles como cadenas.
3. Elija Subscribe.
4. Elija Publicar en un tema, introduzca **`/resnet-50/test`** como Nombre del tema y seleccione Publicar.
5. Si la prueba se realiza correctamente, el mensaje publicado hace que la cámara de Raspberry Pi capture una imagen. Un mensaje de la función de Lambda aparece en la parte inferior de la página. Este mensaje contiene el resultado de predicción de la imagen, con este formato: nombre de la clase predicha, probabilidad y uso máximo de memoria.

Configuración de Intel Atom

Para ejecutar este tutorial en un dispositivo Intel Atom, debe proporcionar imágenes de origen, configurar la función de Lambda y agregar otro recurso del dispositivo local. Para utilizar la GPU como inferencia, compruebe que el siguiente software está instalado en el dispositivo:

- OpenCL versión 1.0 o posterior
- Python 3.7 y pip
- NumPy
- OpenCV en Wheels

1. Descargue imágenes PNG o JPG estáticas para la función de Lambda que se va a utilizar para la clasificación de imágenes. El ejemplo funciona mejor con archivos de imágenes pequeños.

   Guarde los archivos de imágenes en el directorio que contiene el archivo `inference.py` (o en un subdirectorio de este directorio). Se encuentra en el paquete de implementación de la función de Lambda que cargó en the section called “Crear una función de Lambda de inferencia.”.

   > **Note**
   >
   > Si está utilizando AWS DeepLens, puede emplear la cámara integrada o montar su propia cámara para realizar la inferencia de imágenes capturadas en lugar de imágenes

estáticas. Sin embargo, le recomendamos encarecidamente que empiece primero con imágenes estáticas.
Si utiliza una cámara, asegúrese de que el paquete `awscam` de APT esté instalado y actualizado. Para obtener más información, consulte Actualización del dispositivo AWS DeepLens en la Guía para desarrolladores de AWS DeepLens.

2. Edite la configuración de la función de Lambda. Siga el procedimiento indicado en the section called “Añadir la función de Lambda al grupo”.

> **Note**
>
> Le recomendamos que ejecute la función de Lambda sin creación de contenedores, a menos que su modelo de negocio lo requiera. Esto permite el acceso a la GPU y la cámara del dispositivo sin necesidad de configurar los recursos del dispositivo.
> Si ejecuta sin creación de contenedores, también debe conceder acceso raíz a las funciones de Lambda AWS IoT Greengrass.

a. Para ejecutar sin creación de contenedores:

- En Usuario y grupo del sistema, elija **`Another user ID/group ID`**. En ID de usuario del sistema, introduzca **`0`**. Para el ID de grupo del sistema, introduzca **`0`**.

  Esto permite que la función de Lambda se ejecute como raíz. Para obtener más información sobre cómo ejecutar un trabajo, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

  > **Tip**
  >
  > También debe actualizar el archivo `config.json` para conceder acceso raíz a la función de Lambda. Para informarse sobre este procedimiento, consulte the section called “Ejecución de una función de Lambda como raíz”.

- Para la creación de contenedores de la función de Lambda, elija Sin contenedor.

  Para obtener más información sobre la ejecución sin creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.

- Aumente el valor de Timeout (Tiempo de espera) a 2 minutos. De este modo, se asegurará de que la solicitud no supere el tiempo de inactividad demasiado pronto. La configuración tarda unos minutos en ejecutar la inferencia.
- En Ancladas, elija Verdadero
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.

b. Para ejecutarlo en modo contenerizado, en su lugar:

> **Note**
>
> No recomendamos ejecutarlo en modo contenerizado a menos que su modelo de negocio lo requiera.

- Aumente el valor de Memory limit (Límite de memoria) a 3000 MB.
- Aumente el valor de Timeout (Tiempo de espera) a 2 minutos. De este modo, se asegurará de que la solicitud no supere el tiempo de inactividad demasiado pronto. La configuración tarda unos minutos en ejecutar la inferencia.
- En Ancladas, elija Verdadero
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.

3. Agregar al grupo un recurso del modelo optimizado para Neo Cargue los recursos del modelo en el directorio `resnet50` del paquete de ejemplo en el que descomprimió the section called “Crear una función de Lambda de inferencia.”. Este directorio contiene artefactos de modelos compilados previamente para un modelo de clasificación de imágenes entrenado con Resnet-50. Utilice el procedimiento que se describe en the section called “Agregar al grupo un recurso del modelo optimizado para Neo” con las siguientes actualizaciones.

   - Comprima los archivos que están en el directorio `resnet50` en un archivo llamado `resnet50.zip`.
   - En la página Create a machine learning resource (Crear un recurso de aprendizaje automático), en Resource name (Nombre del recurso), escriba **`resnet50_model`**.
   - Cargue el archivo `resnet50.zip`.

4. Si se ejecuta en modo contenerizado, añada el recurso de dispositivo local necesario para conceder acceso a su GPU de dispositivo.

> **Note**
>
> Si lo ejecuta en modo no contenerizado, AWS IoT Greengrass podrá acceder a la GPU del dispositivo sin necesidad de configurar este recurso del dispositivo.

a. En la página de configuración del grupo, elija la pestaña Recursos.

b. En la pestaña Recursos locales, elija Añadir recurso local.

c. Defina el recurso:

- En Nombre del recurso, escriba **`renderD128`**.
- En Tipo de recurso, elija Dispositivo.
- En Ruta del dispositivo, escriba **`/dev/dri/renderD128`**.
- En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.

Configuración de un dispositivo NVIDIA Jetson TX2

Para ejecutar este tutorial en NVIDIA Jetson TX2, proporcione imágenes de origen, configure la función de Lambda y agregue más recursos de dispositivos locales.

1. Compruebe que el dispositivo Jetson está configurado de forma que pueda instalar el software AWS IoT Greengrass Core y utilizar la GPU para la inferencia. Para obtener más información sobre la configuración del proyecto, consulte the section called “Configuración de otros dispositivos”. Si desea utilizar la GPU para realizar la inferencia de un dispositivo NVIDIA Jetson TX2, debe instalar CUDA 10.0 y CUDNN 7.0 en el dispositivo cuando cree una imagen de la placa con Jetpack 4.3.
2. Descargue imágenes PNG o JPG estáticas para la función de Lambda que se va a utilizar para la clasificación de imágenes. El ejemplo funciona mejor con archivos de imágenes pequeños.

   Guarde los archivos de la imagen en la carpeta que contiene el archivo `inference.py`. También puede guardarlos en un subdirectorio de esta carpeta. Este directorio esté en el paquete de implementación de funciones de Lambda que ha cargado en the section called “Crear una función de Lambda de inferencia.”.

> **Note**
>
> En su lugar, puede elegir instrumentar una cámara en la placa Jetson para capturar las imágenes de origen. Sin embargo, le recomendamos encarecidamente que empiece primero con imágenes estáticas.

3. Edite la configuración de la función de Lambda. Siga el procedimiento indicado en the section called “Añadir la función de Lambda al grupo”.

> **Note**
>
> Le recomendamos que ejecute la función de Lambda sin creación de contenedores, a menos que su modelo de negocio lo requiera. Esto permite el acceso a la GPU y la cámara del dispositivo sin necesidad de configurar los recursos del dispositivo. Si ejecuta sin creación de contenedores, también debe conceder acceso raíz a las funciones de Lambda AWS IoT Greengrass.

   a. Para ejecutar sin creación de contenedores:

      - En Ejecutar como, elija **`Another user ID/group ID`**. En UID, escriba **`0`**. En GUID, escriba **`0`**.

        Esto permite que la función de Lambda se ejecute como raíz. Para obtener más información sobre cómo ejecutar un trabajo, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

        > **Tip**
        >
        > También debe actualizar el archivo `config.json` para conceder acceso raíz a la función de Lambda. Para informarse sobre este procedimiento, consulte the section called “Ejecución de una función de Lambda como raíz”.

      - Para la creación de contenedores de la función de Lambda, elija Sin contenedor.

        Para obtener más información sobre la ejecución sin creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.

- Aumente el valor de Timeout (Tiempo de espera) a 5 minutos. De este modo, se asegurará de que la solicitud no supere el tiempo de inactividad demasiado pronto. La configuración tarda unos minutos en ejecutar la inferencia.
- En Ancladas, elija Verdadero
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.

b. Para ejecutarlo en modo contenerizado, en su lugar:

> **Note**
>
> No recomendamos ejecutarlo en modo contenerizado a menos que su modelo de negocio lo requiera.

- Aumente el valor de Memory limit (Límite de memoria). Para utilizar el modelo proporcionado en modo de GPU, utilice 2000 MB.
- Aumente el valor de Timeout (Tiempo de espera) a 5 minutos. De este modo, se asegurará de que la solicitud no supere el tiempo de inactividad demasiado pronto. La configuración tarda unos minutos en ejecutar la inferencia.
- En Ancladas, elija Verdadero
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.

4. Agregar al grupo un recurso del modelo optimizado para Neo Cargue los recursos del modelo en el directorio `resnet18` del paquete de ejemplo en el que descomprimió the section called “Crear una función de Lambda de inferencia.”. Este directorio contiene artefactos precompilados de un modelo de clasificación de imágenes entrenado con Resnet-18. Utilice el procedimiento que se describe en the section called “Agregar al grupo un recurso del modelo optimizado para Neo” con las siguientes actualizaciones.

   - Comprima los archivos que están en el directorio `resnet18` en un archivo llamado `resnet18.zip`.
   - En la página Create a machine learning resource (Crear un recurso de aprendizaje automático), en Resource name (Nombre del recurso), escriba **`resnet18_model`**.
   - Cargue el archivo `resnet18.zip`.

5. Si se ejecuta en modo contenerizado, añada los recursos de dispositivo local necesario para conceder acceso a su GPU de dispositivo.

> **Note**
>
> Si lo ejecuta en modo no contenerizado, AWS IoT Greengrass podrá acceder a la GPU del dispositivo sin necesidad de configurar este recurso del dispositivo.

a. En la página de configuración del grupo, elija la pestaña Recursos.

b. En la pestaña Recursos locales, elija Añadir recurso local.

c. Defina cada recurso:

- En Resource name (Nombre de recurso) y Device path (Ruta del dispositivo), utilice los valores en la siguiente tabla. Cree un recurso de dispositivo para cada fila de la tabla.
- En Tipo de recurso, elija Dispositivo.
- En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.

Nombre	Ruta del dispositivo
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

6. Si se ejecuta en modo contenerizado, añada el siguiente recurso de volumen local para conceder acceso a la cámara de su dispositivo. Siga el procedimiento indicado en the section called “Agregar al grupo un recurso del modelo optimizado para Neo”.

   > **Note**
   >
   > Si lo ejecuta en modo no contenerizado, AWS IoT Greengrass podrá acceder a la cámara del dispositivo sin necesidad de configurar este recurso del dispositivo.

   - En Resource type (Tipo de recurso), elija Device (Dispositivo).
   - En Propietario del grupo del sistema y permisos de acceso a archivos, seleccione Añadir automáticamente permisos del sistema de archivos del grupo del sistema propietario del recurso.

Nombre	Ruta de origen	Destination path
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

7. Actualice las suscripciones del grupo para utilizar el directorio correcto. Utilice el procedimiento que se describe en the section called “Agregar suscripciones al grupo” con las siguientes actualizaciones.

   - En el primer filtro de tema, escriba **`/resnet-18/predictions`**.
   - En el segundo filtro de tema, escriba **`/resnet-18/test`**.

8. Actualice las suscripciones de prueba para utilizar el directorio correcto. Utilice el procedimiento que se describe en the section called “Prueba del ejemplo” con las siguientes actualizaciones.

   - Para Suscripciones, seleccione Suscribirse a un tema. Para Tema de suscripción, escriba **`/resnet-18/predictions`**.
   - En la página `/resnet-18/predictions`, especifique el tema `/resnet-18/test` en el que se va a publicar.

Solución de problemas relacionados con la inferencia de machine learning de AWS IoT Greengrass

Si la prueba no se realiza correctamente, puede seguir estos pasos de solución de problemas. Ejecute los comandos en el terminal de Raspberry Pi.

Comprobación de los registros de error

1. Cambie al usuario raíz y vaya al directorio `log`. El acceso a los registros de AWS IoT Greengrass requiere permisos raíz.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. Compruebe `runtime.log` para ver si hay errores.

   ```
   cat system/runtime.log | grep 'ERROR'
   ```

   También puede consultar el registro de la función de Lambda definida por el usuario para ver si hay errores:

   ```
   cat user/your-region/your-account-id/lambda-function-name.log | grep 'ERROR'
   ```

   Para obtener más información, consulte the section called “Solución de problemas con los registros”.

Verificación de que la función de Lambda se ha implementado correctamente

1. Lista el contenido de la Lambda desplegada en el directorio `/lambda`. Reemplace los valores de los marcadores de posición antes de ejecutar el comando .

   ```
   cd /greengrass/ggc/deployment/lambda/
   arn:aws:lambda:region:account:function:function-name:function-version
   ls -la
   ```

2. Compruebe que el directorio contiene el mismo contenido que el paquete de implementación `optimizedImageClassification.zip` que cargó en el Paso 3: Crear una función de Lambda de inferencia.

   Asegúrese también de que los archivos `.py` y las dependencias se encuentran en la raíz del directorio.

Verificación de que el modelo de inferencia se ha implementado correctamente

1. Busque el número de identificación de proceso (PID) del proceso de tiempo de ejecución de Lambda:

   ```
   ps aux | grep lambda-function-name
   ```

   En el resultado, el PID aparece en la segunda columna de la línea correspondiente al proceso de tiempo de ejecución de Lambda.

2. Entre en el espacio de nombres del tiempo de ejecución de Lambda. Asegúrese de que sustituye el valor del marcador de posición *pid* antes de ejecutar el comando.

   > **Note**
   >
   > Este directorio y su contenido se encuentran en el espacio de nombres del tiempo de ejecución de Lambda, por lo que no son visibles en un espacio de nombres de Linux normal.

   ```
   sudo nsenter -t pid -m /bin/bash
   ```

3. Muestre el contenido del directorio local que ha especificado para el recurso de machine learning.

   > **Note**
   >
   > Si su ruta de recurso de machine learning es distinta de `ml_model`, debe sustituirla aquí.

```
cd /ml_model
ls -ls
```

Debería ver los siguientes archivos:

```
    56 -rw-r--r-- 1 ggc_user ggc_group     56703 Oct 29 20:07 model.json
196152 -rw-r--r-- 1 ggc_user ggc_group 200855043 Oct 29 20:08 model.params
   256 -rw-r--r-- 1 ggc_user ggc_group    261848 Oct 29 20:07 model.so
    32 -rw-r--r-- 1 ggc_user ggc_group     30564 Oct 29 20:08 synset.txt
```

La función de Lambda no puede encontrar **/dev/dri/renderD128**

Esto puede ocurrir si OpenCL no puede conectarse a los dispositivos de GPU que necesita. Debe crear los recursos de dispositivos para los dispositivos necesarios para su función de Lambda.

Pasos siguientes

A continuación, explore otros modelos optimizados. Para obtener más información, consulte la documentación de SageMaker Neo.

Administrar secuencias de datos en el núcleo de AWS IoT Greengrass

El administrador de secuencias AWS IoT Greengrass hace que sea más fácil y confiable transferir datos de IoT de gran volumen a la Nube de AWS. Un administrador de secuencias procesa flujos de datos localmente y los exporta automáticamente a la Nube de AWS. Esta característica se integra con situaciones límite habituales, como la inferencia de aprendizaje automático (ML), donde los datos se procesan y analizan localmente antes de exportarlos a la Nube de AWS o a destinos de almacenamiento local.

El administrador de secuencias simplifica el desarrollo de aplicaciones. Sus aplicaciones de IoT pueden utilizar un mecanismo estandarizado para procesar secuencias de gran volumen y administrar políticas de retención de datos locales en lugar de crear funciones de administración de secuencias personalizadas. Las aplicaciones de IoT pueden leer y escribir en secuencias. Pueden definir políticas para el tipo de almacenamiento, el tamaño y la retención de datos en función de cada secuencia para controlar cómo el administrador de secuencias procesa y exporta secuencias.

El administrador de secuencias está diseñado para trabajar en entornos con conectividad intermitente o limitada. Puede definir el uso del ancho de banda, el comportamiento del tiempo de espera y cómo se administran los datos de secuencias cuando el núcleo está conectado o desconectado. Para los datos críticos, puede establecer prioridades para controlar el orden en que se exportan las secuencias a la Nube de AWS.

Puede configurar las exportaciones automáticas a la Nube de AWS para su almacenamiento y posterior procesamiento y análisis. El administrador de secuencias admite la exportación a los siguientes destinos Nube de AWS.

- Canales en AWS IoT Analytics. AWS IoT Analytics le permite realizar análisis avanzados de sus datos para ayudarle a tomar decisiones empresariales y mejorar los modelos de machine learning. Para obtener más información, consulte ¿Qué es AWS IoT Analytics? en la Guía del usuario de AWS IoT Analytics.
- Fuye por Kinesis Data Streams Kinesis Data Streams se suele utilizar para agregar grandes volúmenes de datos y cargarlos en un almacenamiento de datos o en un clúster de map-reduce. Para obtener más información, consulte ¿Qué son los Amazon Kinesis Data Streams? en la Guía para desarrolladores de Amazon Kinesis.

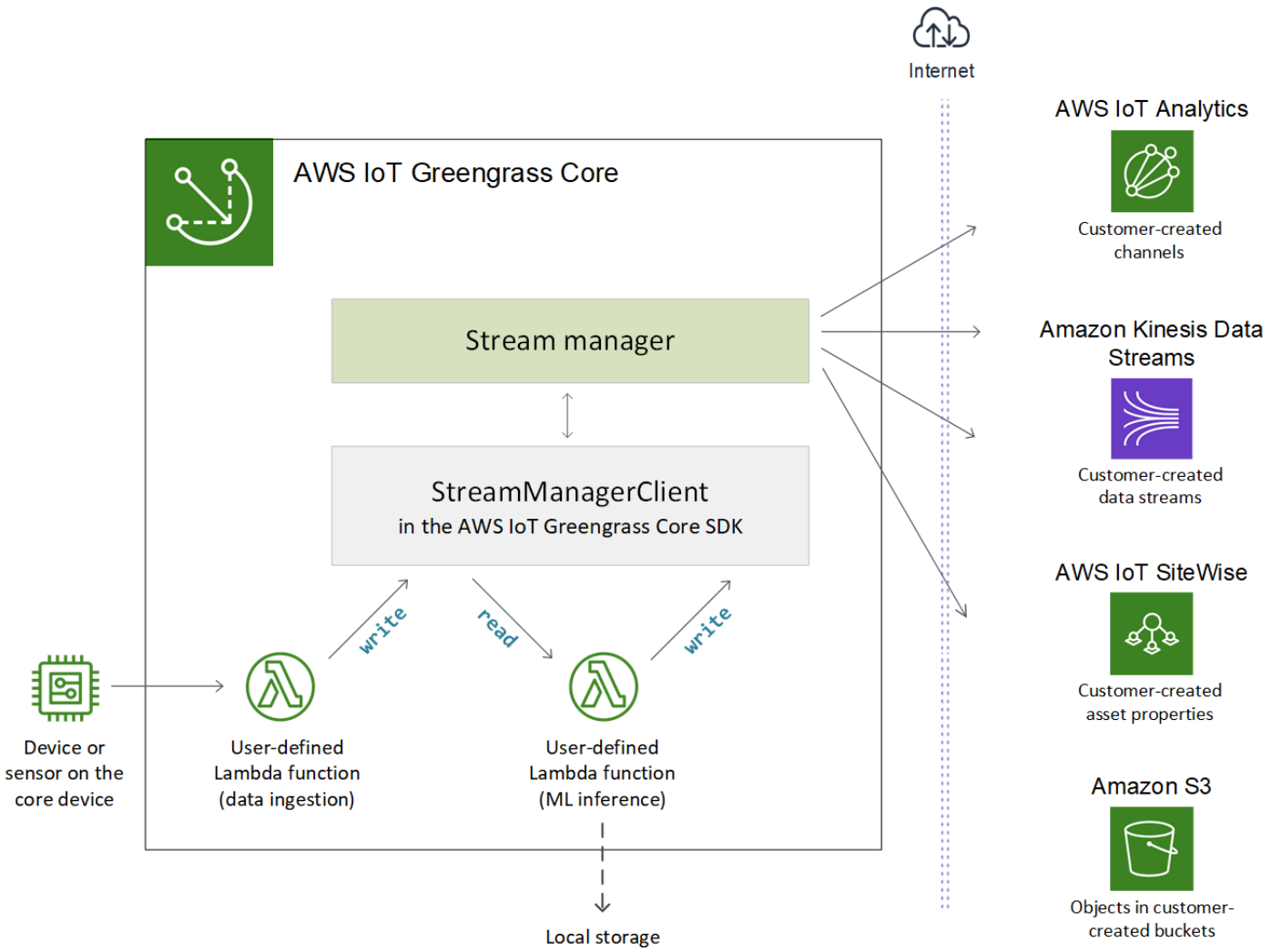
- Propiedades de los activos en AWS IoT SiteWise. AWS IoT SiteWise le permite recopilar, organizar y analizar datos de equipos industriales a escala. Para obtener más información, consulte ¿Qué es AWS IoT SiteWise? en la Guía del usuario de AWS IoT SiteWise.
- Objetos en Amazon S3. Puede utilizar Amazon S3 para almacenar y recuperar grandes cantidades de datos. Para obtener más información, consulte ¿Qué es Amazon S3? en la Guía para desarrolladores de Amazon Simple Storage Service.

Flujo de trabajo de la administración de secuencias

Sus aplicaciones de IoT interactúan con el administrador de flujo a través del SDK de AWS IoT Greengrass Core.. En un flujo de trabajo simple, una función de Lambda definida por el usuario que se ejecuta en el núcleo de Greengrass consume los datos de IoT, como las métricas de temperatura y presión de serie temporal. La función de Lambda podría filtrar o comprimir los datos y luego llamar al SDK de AWS IoT Greengrass Core para escribir los datos en una secuencia en el administrador de secuencias. El administrador de secuencias puede exportar la secuencia a la Nube de AWS automáticamente, en función de las políticas definidas para la secuencia. Las funciones de Lambda definidas por el usuario también pueden enviar datos directamente a bases de datos locales o repositorios de almacenamiento.

Sus aplicaciones de IoT pueden incluir múltiples funciones de Lambda definidas por el usuario que leen o escriben en secuencias. Estas funciones de Lambda locales pueden leer y escribir en secuencias para filtrar, añadir y analizar datos localmente. Esto permite responder rápidamente a eventos locales y extraer información valiosa antes de que los datos se transfieran desde el núcleo a la nube o destinos locales.

En el siguiente diagrama se muestra un flujo de trabajo de ejemplo.



Para usar el administrador de secuencias, comience por configurar los parámetros del administrador de secuencias para definir los ajustes de tiempo de ejecución a nivel de grupo que se apliquen a todas las secuencias del núcleo de Greengrass. Esta configuración personalizable le permite controlar cómo el administrador de secuencias almacena, procesa y exporta secuencias en función de las necesidades de su negocio y las restricciones del entorno. Para obtener más información, consulte the section called “Configurar el administrador de secuencias”.

Después de configurar el administrador de secuencias, puede crear e implementar sus aplicaciones de IoT. Por lo general, se trata de funciones de Lambda definidas por el usuario que utilizan `StreamManagerClient` en el SDK de AWS IoT Greengrass Core para crear secuencias e interactuar con ellas. Durante la creación de la secuencia, la función de Lambda define las políticas por secuencia, como los destinos de exportación, la prioridad y la persistencia. Para obtener más información, incluidos fragmentos de código para las operaciones `StreamManagerClient`, consulte the section called “Utilizar StreamManagerClient para trabajar con secuencias”.

Para ver tutoriales que configuran un flujo de trabajo sencillo, consulte the section called “Exportar flujos de datos (consola)” o the section called “Exportar secuencias de datos (CLI)”.

Requisitos

Se aplican los siguientes requisitos para el administrador de flujo:

- Debe utilizar el software AWS IoT Greengrass Core versión 1.10 o posterior, con el administrador de secuencias habilitado. Para obtener más información, consulte the section called “Configurar el administrador de secuencias”.

  El administrador de secuencias no es compatible con las distribuciones OpenWrt.
- Debe instalarse Java 8 Runtime (JDK 8) en el núcleo.
  - Para distribuciones basadas en Debian (incluido Raspbian) o distribuciones basadas en Ubuntu, ejecute el siguiente comando:

    ```
    sudo apt install openjdk-8-jdk
    ```
  - Para distribuciones basadas en Red Hat (incluido Amazon Linux), ejecute el siguiente comando:

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Para obtener más información, consulte How to download and install prebuilt OpenJDK packages (Cómo descargar e instalar paquetes OpenJDK preconfigurados) en la documentación de OpenJDK.
- El administrador de secuencias requiere un mínimo de 70 MB de RAM además de su software básico Core AWS IoT Greengrass. El requisito total de memoria depende de la carga de trabajo.
- Las funciones de Lambda definidas por el usuario deben utilizar el SDK de AWS IoT GreengrassCore para interactuar con el administrador de flujo. El SDK AWS IoT Greengrass Core está disponible en varios idiomas, pero solo las siguientes versiones admiten las operaciones del administrador de secuencias.
  - SDK de Java (versión 1.4.0 o posterior)
  - SDK de Python (versión 1.5.0 o posterior)

- SDK de Node.js (versión 1.6.0 o posterior)

  Descargue la versión del SDK que corresponda al tiempo de ejecución de la función de Lambda e inclúyala en el paquete de implementación de la función de Lambda.

  > **Note**
  >
  > El SDK de AWS IoT Greengrass Core para Python requiere Python 3.7 o posterior y tiene otras dependencias de paquetes. Para obtener más información, consulte Crear un paquete de implementación de funciones de Lambda (consola) o Crear un paquete de implementación de funciones de Lambda (CLI).

- Si define destinos de exportación de Nube de AWS para un flujo, deberá crear sus destinos de exportación y conceder permisos de acceso en el rol de grupo Greengrass. Según el destino, es posible que también se apliquen otros requisitos. Para obtener más información, consulte:
  - the section called “Canales de AWS IoT Analytics”
  - the section called “Amazon Kinesis Data Streams”
  - the section called “AWS IoT SiteWise propiedades de activos”
  - the section called “Objetos de Amazon S3”

  Usted es responsable del mantenimiento de estos recursos de Nube de AWS.

Seguridad de los datos

Cuando utilice el administrador de secuencias, tenga en cuenta las siguientes consideraciones de seguridad.

Seguridad de los datos locales

AWS IoT Greengrass no cifra los datos de secuencias en reposo o en tránsito localmente entre los componentes del dispositivo del núcleo.

- Datos en reposo. Los datos de secuencias se almacenan localmente en un directorio de almacenamiento en el núcleo de Greengrass. Para la seguridad de los datos, AWS IoT Greengrass se basa en los permisos de archivo Unix y el cifrado de disco completo, si está habilitado. Puede utilizar el parámetro opcional STREAM_MANAGER_STORE_ROOT_DIR para especificar el directorio de almacenamiento. Si cambia este parámetro más adelante para utilizar un directorio

de almacenamiento diferente, AWS IoT Greengrass no eliminará el directorio de almacenamiento anterior ni su contenido.

- Datos en tránsito localmente. AWS IoT Greengrass no cifra datos de secuencias en tránsito en el núcleo local entre orígenes de datos, funciones de Lambda, el SDK de AWS IoT Greengrass Core y el administrador de flujos.

- Datos en tránsito a la Nube de AWS. Las secuencias de datos exportadas por el administrador de secuencias a la Nube de AWS utilizan cifrado de cliente de servicio AWS estándar con seguridad de la capa de transporte (TLS)

Para obtener más información, consulte the section called “Cifrado de datos”.

Autenticación del cliente

Los clientes del administrador de secuencias utilizan el SDK de AWS IoT Greengrass Core para comunicarse con el administrador de flujos. Cuando la autenticación de cliente está habilitada, sólo las funciones de Lambda del grupo Greengrass pueden interactuar con las secuencias en el administrador de secuencias. Cuando la autenticación de cliente está deshabilitada, cualquier proceso que se ejecute en el núcleo de Greengrass (como contenedores Docker) puede interactuar con las secuencias en el administrador de secuencias. Debe deshabilitar la autenticación solo si su caso de negocio lo requiere.

Utilice el parámetro STREAM_MANAGER_AUTHENTICATE_CLIENT para establecer el modo de autenticación del cliente. Puede configurar este parámetro desde la consola o API de AWS IoT Greengrass. Los cambios surten efecto después de implementar el grupo.

	Habilitado	Deshabilitado
Valor del parámetro	`true` (predeterminado y recomendado)	`false`
Clientes permitidos	Funciones de Lambda definidas por el usuario en el grupo Greengrass	Funciones de Lambda definidas por el usuario en el grupo Greengrass

	Habilitado	Deshabilitado
		Otros procesos que se ejecutan en el dispositivo del núcleo de Greengrass

Véase también

- the section called “Configurar el administrador de secuencias”
- the section called “Utilizar StreamManagerClient para trabajar con secuencias”
- the section called “Exportación de configuraciones para Nube de AWS destinos compatibles”
- the section called “Exportar flujos de datos (consola)”
- the section called “Exportar secuencias de datos (CLI)”

Configurar el administrador de secuencias de AWS IoT Greengrass

En el núcleo AWS IoT Greengrass, el administrador de secuencias puede almacenar, procesar y exportar datos enviados desde dispositivos IoT. El administrador de secuencias proporciona parámetros que se utilizan para configurar los ajustes de tiempo de ejecución en el nivel de grupo. Esta configuración se aplica a todas las secuencias del núcleo de Greengrass. Puede utilizar la consola AWS IoT o la API AWS IoT Greengrass para configurar los ajustes del administrador de secuencias. Los cambios surten efecto después de implementar el grupo.

> **Note**
>
> Después de configurar el administrador de secuencias, puede crear e implementar aplicaciones de IoT que se ejecuten en el núcleo de Greengrass e interactúen con el administrador de secuencias. Estas aplicaciones de IoT suelen ser funciones de Lambda definidas por el usuario. Para obtener más información, consulte the section called “Utilizar StreamManagerClient para trabajar con secuencias”.

Parámetros del administrador de secuencias

El administrador de secuencias proporciona los siguientes parámetros que le permiten definir la configuración en el nivel de grupo. Todos los parámetros son opcionales.

Directorio de almacenamiento

Nombre del parámetro: `STREAM_MANAGER_STORE_ROOT_DIR`

La ruta absoluta del directorio local utilizado para almacenar secuencias. Este valor debe comenzar con una barra inclinada (por ejemplo, `/data`).

Para obtener información sobre cómo proteger los datos de secuencias, consulte the section called “Seguridad de los datos locales”.

Versión mínima AWS IoT Greengrass de Core: 1.10.0

Server port

Nombre del parámetro: `STREAM_MANAGER_SERVER_PORT`

El número de puerto local utilizado para comunicarse con el administrador de secuencias. El valor predeterminado es `8088`.

Versión mínima AWS IoT Greengrass de Core: 1.10.0

Autenticar cliente

Nombre del parámetro: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

Indica si los clientes deben autenticarse para interactuar con el administrador de secuencias. Toda la interacción entre los clientes y el administrador de secuencias está controlada por el SDK AWS IoT Greengrass Core. Este parámetro determina qué clientes pueden llamar al SDK AWS IoT Greengrass Core para trabajar con secuencias. Para obtener más información, consulte the section called “Autenticación del cliente”.

Los valores válidos son `true` o `false`. El valor predeterminado es `true` (recomendado).

- `true`. Solo permite como clientes las funciones de Lambda de Greengrass. Los clientes de funciones de Lambda utilizan protocolos AWS IoT Greengrass principales internos para autenticarse con el SDK de AWS IoT Greengrass Core.
- `false`. Permite que cualquier proceso que se ejecute en el AWS IoT Greengrass core sea un cliente. No establezca a `false` a menos que su caso de negocio lo requiera. Por ejemplo,

establezca este valor en `false` solo si los procesos no Lambda en el dispositivo principal deben comunicarse directamente con el administrador de secuencias, como los contenedores de Docker que se ejecutan en el núcleo.

Versión mínima AWS IoT Greengrass de Core: 1.10.0

Ancho de banda máximo

Nombre del parámetro: `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

El ancho de banda máximo promedio (en kilobits por segundo) que se puede utilizar para exportar datos. El valor predeterminado permite el uso ilimitado del ancho de banda disponible.

Versión mínima AWS IoT Greengrass de Core: 1.10.0

Tamaño del grupo de subprocesos

Nombre del parámetro: `STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

Cantidad máxima de subprocesos activos que se pueden utilizar para exportar datos. El valor predeterminado es `5`.

El tamaño óptimo depende del hardware, el volumen de secuencias y la cantidad planificada de secuencias de exportación. Si la velocidad de exportación es lenta, puede ajustar esta configuración para encontrar el tamaño óptimo para su hardware y su caso de negocio. La CPU y la memoria del hardware del dispositivo principal son factores limitantes. Para comenzar, puede intentar establecer este valor igual a la cantidad de núcleos de procesador en el dispositivo.

Tenga cuidado de no establecer un tamaño superior al que admite el hardware. Cada secuencia consume recursos de hardware, por lo que debe intentar limitar la cantidad de secuencias de exportación en dispositivos restringidos.

Versión mínima AWS IoT Greengrass de Core: 1.10.0

Argumentos de JVM

Nombre del parámetro: `JVM_ARGS`

Argumentos personalizados de la máquina virtual de Java para pasar al administrador de secuencias al inicio. Varios argumentos deben separarse por espacios.

Utilice este parámetro sólo cuando deba anular la configuración predeterminada utilizada por la JVM. Por ejemplo, puede que necesite aumentar el tamaño predeterminado del montón si planea exportar un gran número de secuencias.

Versión mínima AWS IoT Greengrass de Core: 1.10.0

Directorios de archivos de entrada de solo lectura

Nombre del parámetro: `STREAM_MANAGER_READ_ONLY_DIRS`

Lista de rutas absolutas separadas por comas a los directorios situados fuera del sistema de archivos raíz que almacenan los archivos de entrada. El administrador de secuencias lee y carga los archivos en Amazon S3 y monta los directorios como de solo lectura. Para obtener más información sobre la exportación a Amazon S3, consulte the section called “Objetos de Amazon S3”.

Utilice este parámetro solo si se cumplen las siguientes condiciones:

- El directorio de archivos de entrada de una transmisión que se exporta a Amazon S3 se encuentra en una de las siguientes ubicaciones:
  - Una partición distinta del sistema de archivos raíz.
  - En `/tmp` el sistema de archivos raíz.
- La creación de contenedores por defecto del grupo Greengrass es contenedor de Greengrass.

Ejemplo de valor: `/mnt/directory-1,/mnt/directory-2,/tmp`

Versión mínima AWS IoT Greengrass de Core: 1.11.0

Tamaño mínimo para la carga de varias partes

Nombre del parámetro: `STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

El tamaño mínimo (en bytes) de una parte en una carga multiparte a Amazon S3. El administrador de secuencias utiliza esta configuración y el tamaño del archivo de entrada para determinar cómo agrupar los datos en una solicitud PUT de varias partes. El valor por defecto y mínimo es de `5242880` bytes (5 MB).

> **Note**
>
> El administrador de secuencias usa la propiedad `sizeThresholdForMultipartUploadBytes` de la transmisión para determinar si se debe exportar a Amazon S3 como una carga única o multiparte. Las funciones de Lambda definidas por el usuario establecen este umbral cuando crean una transmisión que se exporta a Amazon S3. El umbral de tamaño predeterminado es 5 MB.

Versión mínima AWS IoT Greengrass de Core: 1.11.0

Configuración del administrador de secuencias (consola)

Puede utilizar la consola AWS IoT para las siguientes tareas de administración:

- Comprobar si el administrador de secuencias está habilitado
- Habilitar o deshabilitar el administrador de secuencias durante la creación de grupos
- Habilitar o deshabilitar el administrador de secuencias en un grupo existente
- Cambiar la configuración del administrador de secuencias

Los cambios surten efecto después de que se implemente el grupo de Greengrass. Para ver un tutorial que muestra cómo implementar un grupo de Greengrass que contiene una función de Lambda que interactúa con el administrador de flujos, consulte the section called “Exportar flujos de datos (consola)”.

> **Note**
>
> Cuando utiliza la consola para habilitar el administrador de secuencias y desplegar el grupo, el tamaño de la memoria para el administrador de transmisiones se establece por defecto en 4194304 KB (4 GB). Se recomienda establecer el tamaño de la memoria en al menos 128000 KB.

Para comprobar si el administrador de secuencias está habilitado (consola)

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. Elija la pestaña funciones de Lambda.
4. En Funciones de Lambda del sistema, seleccione Administrador de secuencias y luego elija Editar.
5. Compruebe el estado activado o desactivado. También se muestra cualquier ajuste personalizado del administrador de secuencias que se haya configurado.

Para habilitar o deshabilitar el administrador de secuencias durante la creación de grupos (consola)

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija Create Group. Su elección en la página siguiente determina cómo configurar el administrador de secuencias para el grupo.
3. Proceda a través de las páginas Nombre su grupo y elija un núcleo Greengrass.
4. Elija Create group.
5. En la página de configuración del grupo, elija la pestaña Funciones de Lambda, seleccione Administrador de secuencias y elija Editar.

   - Para habilitar el administrador de secuencias con la configuración predeterminada, elija Habilitar valores predeterminados.

   - Para habilitar el administrador de secuencias con configuración personalizada, elija Customize settings (Personalizar configuración).
     1. En la página Configurar administrador de secuencias elija Habilitar.
     2. En Custom settings (Configuración personalizada), introduzca valores para los parámetros del administrador de secuencias. Para obtener más información, consulte the section called “Parámetros del administrador de secuencias”. Deje los campos vacíos para permitir que AWS IoT Greengrass use sus valores predeterminados.

   - Para deshabilitar el administrador de secuencias, seleccione Desactivar.
     1. En la página Configure stream manager (Configurar administrador de secuencias) elija Disable (Deshabilitar).

6. Seleccione Save.
7. Continúe en las páginas restantes para crear su grupo.
8. En la página Dispositivos cliente descargue los recursos de seguridad, revise la información y, a continuación, elija Finalizar.

> **Note**
>
> Cuando el administrador de secuencias esté habilitado, debe instalar Java 8 Runtime en el dispositivo principal antes de implementar el grupo.

Para habilitar o deshabilitar el administrador de secuencias para un grupo existente (consola)

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. Elija la pestaña funciones de Lambda.
4. En Funciones de Lambda del sistema, seleccione Administrador de secuencias y luego elija Editar.
5. Compruebe el estado activado o desactivado. También se muestra cualquier ajuste personalizado del administrador de secuencias que se haya configurado.

Para cambiar la configuración del administrador de secuencias (consola)

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. Elija la pestaña funciones de Lambda.
4. En Funciones de Lambda del sistema, seleccione Administrador de secuencias y luego elija Editar.
5. Compruebe el estado activado o desactivado. También se muestra cualquier ajuste personalizado del administrador de secuencias que se haya configurado.
6. Seleccione Save.

Configuración del administrador de secuencias (CLI)

En la AWS CLI se utiliza la función de Lambda `GGStreamManager` del sistema para configurar el administrador de secuencias. Las funciones de Lambda del sistema son componentes del software de AWS IoT Greengrass Core. Para el administrador de secuencias y algunas otras funciones de Lambda del sistema, puede configurar la funcionalidad de Greengrass gestionando los objetos `Function` y `FunctionDefinitionVersion` correspondientes del grupo Greengrass. Para obtener más información, consulte the section called “Información general sobre el modelo de objetos de grupo”.

Puede utilizar la API para las siguientes tareas de administración. En los ejemplos de esta sección se muestra cómo utilizar la APIAWS CLI, pero también se puede llamar directamente a la AWS IoT Greengrass API o utilizar un SDK de AWS.

- Comprobar si el administrador de secuencias está habilitado
- Habilitar, deshabilitar o configurar el administrador de secuencias

Los cambios surten efecto después de implementar el grupo. Para ver un tutorial que muestra cómo implementar un grupo de Greengrass con una función de Lambda que interactúa con el administrador de secuencias, consulte. the section called “Exportar secuencias de datos (CLI)”

> **Tip**
>
> Para ver si el administrador de secuencias está habilitado y en ejecución desde su dispositivo principal, puede ejecutar el siguiente comando en un terminal del dispositivo principal.
>
> ```
> ps aux | grep -i 'streammanager'
> ```

Para comprobar si el administrador de secuencias está habilitado (CLI)

El administrador de secuencias está habilitado si la versión de definición de característica implementada incluye la función de Lambda del sistema `GGStreamManager`. Para verificar, haga lo siguiente;

1. Obtenga los ID de la versión de grupo y grupo de Greengrass de destino. En este procedimiento, suponemos que estos son el último grupo y la última versión de grupo. La siguiente consulta devuelve el grupo creado más recientemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   También puede hacer la consulta por nombre. No es necesario que los nombres de grupo sean únicos, por lo que podrían devolverse varios grupos.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

2. Copie los valores `Id` y `LatestVersion` del grupo de destino en la salida.
3. Obtenga la última versión del grupo.

   - Reemplace *`id-grupo`* con el `Id` que ha copiado.
   - Reemplace *`id-última-versión-grupo`* con la `LatestVersion` que ha copiado.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id latest-group-version-id
   ```

4. En el objeto `FunctionDefinitionVersionArn` de la salida, obtenga el ID y la versión de la definición de funciones.

   - El ID de definición de característica es el GUID que sigue al segmento `functions` en el Nombre de recurso de Amazon (ARN).
   - El ID de versión de definición de característica es el GUID que sigue al segmento `versions` del ARN.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/function-definition-id/versions/function-definition-version-id
```

5. Obtenga la versión de la definición de funciones.

   - Reemplace *function-definition-id* con el ID de definición de característica.
   - Reemplace *function-definition-version-id* con el ID de versión de definición de característica.

```
aws greengrass get-function-definition-version \
--function-definition-id function-definition-id \
--function-definition-version-id function-definition-version-id
```

Si la matriz `functions` en la salida incluye la característica `GGStreamManager`, entonces el administrador de secuencias está habilitado. Cualquier variable de entorno definida para la característica representa una configuración personalizada para el administrador de secuencias.

Para habilitar, deshabilitar o configurar el administrador de secuencias(CLI)

En la AWS CLI se utiliza la función de Lambda `GGStreamManager` del sistema para configurar el administrador de secuencias. Los cambios surten efecto después de implementar el grupo.

- Para habilitar el administrador de secuencias, incluya `GGStreamManager` en la matriz `functions` de su versión de definición de característica. Para establecer una configuración personalizada, defina variables de entorno para los parámetros del administrador de secuencias correspondientes.
- Para deshabilitar el administrador de secuencias, elimine `GGStreamManager` de la matriz `functions` de su versión de definición de característica.

Administrador de secuencias con una configuración predeterminada

En el siguiente ejemplo de configuración se habilita el administrador de secuencias con una configuración predeterminada. Establece el ID de característica arbitraria en `streamManager`.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
```

```
    "FunctionConfiguration": {
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

> **Note**
>
> En cuanto a las propiedades `FunctionConfiguration`, es posible que sepa lo siguiente:
>
> - `MemorySize` está establecido en 4194304 KB (4 GB) con la configuración predeterminada. Siempre puede cambiar este valor. Se recomienda establecer `MemorySize` en al menos 128000 KB.
> - `Pinned` se debe establecer en `true`.
> - `Timeout` es necesario para la versión de definición de característica, pero `GGStreamManager` no lo usa.

Administrador de secuencias con una configuración personalizada

El siguiente ejemplo de configuración habilita el gestor de flujos con valores personalizados para los parámetros directorio de almacenamiento, puerto del servidor y tamaño del grupo de hilos.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                "STREAM_MANAGER_SERVER_PORT": "1234",
                "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
            }
        },
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
```

```
}
```

AWS IoT Greengrass utiliza valores predeterminados para los parámetros del administrador de secuencias que no están especificados como variables de entorno.

El administrador de secuencias con ajustes personalizados para las exportaciones de Amazon S3

El siguiente ejemplo de configuración habilita el administrador de secuencias con valores personalizados para los parámetros directorio de carga y tamaño mínimo de carga multiparte.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_READ_ONLY_DIRS": "/mnt/directory-1,/mnt/
directory-2,/tmp",

 "STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES":
 "10485760"
            }
        },
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

Para habilitar, deshabilitar o configurar el administrador de secuencias (CLI)

1. Obtenga los ID de la versión de grupo y grupo de Greengrass de destino. En este procedimiento, suponemos que estos son el último grupo y la última versión de grupo. La siguiente consulta devuelve el grupo creado más recientemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

También puede hacer la consulta por nombre. No es necesario que los nombres de grupo sean únicos, por lo que podrían devolverse varios grupos.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

> **Note**
>
> También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

2. Copie los valores `Id` y `LatestVersion` del grupo de destino en la salida.
3. Obtenga la última versión del grupo.

   - Reemplace *id-grupo* con el `Id` que ha copiado.
   - Reemplace *id-última-versión-grupo* con la `LatestVersion` que ha copiado.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. Copie el `CoreDefinitionVersionArn` y todos los demás ARN de la versión de la salida, excepto `FunctionDefinitionVersionArn`. Estos valores los utilizará más adelante cuando cree una versión de grupo.
5. Desde `FunctionDefinitionVersionArn` en el resultado, copie el ID de la definición de la característica. El ID es el GUID que va detrás del segmento `functions` en el ARN, tal y como se muestra en el siguiente ejemplo.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> O puede crear una definición de la función ejecutando el comando `create-function-definition` y luego copiar el ID del resultado.

6. Añada una versión de definición de la función a la definición de la característica.

   - Reemplace *`function-definition-id`* con el `Id` que ha copiado de la definición de la función.
   - En la matriz `functions`, incluya todas las demás funciones que desee que estén disponibles en el núcleo de Greengrass. Puede usar el comando `get-function-definition-version` para obtener la lista de funciones existentes.

   Habilitar el administrador de secuencias con una configuración predeterminada

   En el ejemplo siguiente se habilita el administrador de secuencias al incluir la característica `GGStreamManager` en la matriz `functions`. En este ejemplo se utilizan valores predeterminados para los parámetros del administrador de secuencias.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize":  4194304,
                "Pinned": true,
                "Timeout": 3
            },
            "Id": "streamManager"
        },
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
```

```
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> La función `myLambdaFunction` de los ejemplos representa una de las funciones de Lambda que ha definido.

Habilitar el administrador de secuencias con una configuración personalizada

En el ejemplo siguiente se habilita el administrador de secuencias al incluir la característica `GGStreamManager` en la matriz `functions`. Todas las configuraciones del administrador de secuencias son opcionales, a menos que desee cambiar los valores predeterminados. Este ejemplo muestra cómo utilizar variables de entorno para establecer valores personalizados.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "Environment": {
                    "Variables": {
                        "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                        "STREAM_MANAGER_SERVER_PORT": "1234",
                        "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
                    }
                },
                "MemorySize":  4194304,
                "Pinned": true,
                "Timeout": 3
            },
            "Id": "streamManager"
```

```
        },
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> En cuanto a las propiedades `FunctionConfiguration`, es posible que sepa lo siguiente:
>
> - `MemorySize` está establecido en 4194304 KB (4 GB) con la configuración predeterminada. Siempre puede cambiar este valor. Se recomienda establecer `MemorySize` en al menos 128000 KB.
> - `Pinned` se debe establecer en `true`.
> - `Timeout` es necesario para la versión de definición de característica, pero `GGStreamManager` no lo usa.

Desactivar el administrador de secuencias

El ejemplo siguiente se omite la característica `GGStreamManager`, lo que desactiva el administrador de secuencias.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
```

```
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> Si no desea implementar ninguna función de Lambda, puede omitir la versión de la definición de característica por completo.

7. Copie el `Arn` de la versión de definición de la característica del resultado.
8. Cree una versión de grupo que contenga la función de Lambda del sistema.

   - Reemplace *`id-grupo`* con el `Id` del grupo.
   - Reemplace *`arn-versión-definición-núcleo`* con el `CoreDefinitionVersionArn` que ha copiado de la última versión del grupo.
   - Reemplace *`function-definition-version-arn`* con el `Arn` que ha copiado para la nueva versión de definición de la característica.
   - Reemplace los ARN para otros componentes del grupo (por ejemplo, `SubscriptionDefinitionVersionArn` o `DeviceDefinitionVersionArn`) que ha copiado de la última versión del grupo.
   - Elimine los parámetros no utilizados. Por ejemplo, elimine `--resource-definition-version-arn` si la versión de su grupo no contiene ningún recurso.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Copie la `Version` del resultado. Este es el ID de la nueva versión del grupo.
10. Implemente el grupo con la nueva versión del grupo.

    - Reemplace *`id-grupo`* con el `Id` que ha copiado para el grupo.
    - Reemplace *`id-versión-grupo`* con el `Version` que ha copiado para el nuevo grupo.

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

Siga este procedimiento si desea volver a editar la configuración del administrador de secuencias más adelante. Asegúrese de crear una versión de definición de la función que incluya la función `GGStreamManager` con la configuración actualizada. La versión de grupo debe hacer referencia a todos los ARN de las versiones de los componentes que desee implementar en el núcleo. Los cambios surten efecto después de implementar el grupo.

Véase también

- Administrar secuencias de datos
- the section called “Utilizar StreamManagerClient para trabajar con secuencias”
- the section called “Exportación de configuraciones para Nube de AWS destinos compatibles”
- the section called “Exportar flujos de datos (consola)”
- the section called “Exportar secuencias de datos (CLI)”

Utilizar StreamManagerClient para trabajar con secuencias

Las funciones Lambda definidas por el usuario que se ejecutan en el AWS IoT Greengrass pueden utilizar el objeto `StreamManagerClient` en el AWS IoT Greengrass Core SDK para crear e interactuar con secuencias en el administrador de secuencias. Cuando una función Lambda crea una secuencia, define los destinos Nube de AWS de la nube, la priorización y otras políticas de exportación y conservación de datos para la secuencia. Para enviar datos al administrador de flujos,

las funciones de Lambda anexan los datos al flujo. Si se define un destino de exportación para el flujo, el administrador de secuencias exporta la secuencia automáticamente.

> **Note**
>
> Normalmente, los clientes del administrador de secuencias son funciones de Lambda definidas por el usuario. Si su caso de negocio lo requiere, puede permitir que los procesos no-Lambda que se ejecutan en el núcleo de Greengrass (por ejemplo, un contenedor de Docker) interactúen con el administrador de secuencias. Para obtener más información, consulte the section called “Autenticación del cliente”.

Los fragmentos de este tema muestran cómo los clientes llaman `StreamManagerClient` métodos para trabajar con secuencias. Para obtener detalles sobre la implementación de los métodos y sus argumentos, utilice los vínculos a la referencia del SDK de cada fragmento. Para ver tutoriales que utilizan una función the section called “Exportar flujos de datos (consola)” completa de Python, consulte o the section called “Exportar secuencias de datos (CLI)”.

Debe crear una instancia Lambda `StreamManagerClient` fuera del controlador de funciones. Si se crea una instancia en el controlador, la función crea un `client` y una conexión al administrador de secuencias cada vez que se invoca.

> **Note**
>
> Si crea una instancia `StreamManagerClient` en el controlador, debe llamar explícitamente al método `close()` cuando el `client` complete su trabajo. De lo contrario, el `client` mantiene la conexión abierta y otro subproceso en ejecución hasta que salga del script.

`StreamManagerClient` admite las siguientes operaciones:

- the section called “Creación de una secuencia de mensajes”
- the section called “Agregar un mensaje”
- the section called “Lectura de mensajes”
- the section called “Lista de secuencias”
- the section called “Descripción de una secuencia de mensajes”
- the section called “Actualizar una secuencia de mensajes”

- the section called “Eliminación de una secuencia de mensajes”

Creación de una secuencia de mensajes

Para crear un flujo, una función de Lambda definida por el usuario llama al método createcrear y pasa un objeto.objeto`MessageStreamDefinition`. Este objeto especifica el nombre exclusivo del flujo y define cómo el administrador del flujo debe gestionar los nuevos datos cuando se alcanza el tamaño máximo de flujo. Puede utilizar `MessageStreamDefinition` y sus tipos de datos (como `ExportDefinition`, `StrategyOnFull` y `Persistence`) para definir otras propiedades de secuencias. Entre ellas se incluyen:

- El destino AWS IoT Analytics, Kinesis Data Streams, AWS IoT SiteWise y Amazon S3 de destino para las exportaciones automáticas. Para obtener más información, consulte the section called “Exportación de configuraciones para Nube de AWS destinos compatibles”.
- Prioridad de exportación. El administrador de secuencias exporta secuencias de mayor prioridad antes que secuencias de menor prioridad.
- Tamaño e intervalo de lote máximos para AWS IoT Analytics, Kinesis Data Streams y destinos AWS IoT SiteWise. El administrador de secuencias exporta mensajes cuando se cumple cualquiera de las condiciones.
- Time-to-Live (TTL). La cantidad de tiempo para garantizar que los datos de secuencia están disponibles para su procesamiento. Debe asegurarse de que los datos se pueden consumir dentro de este período de tiempo. Esta no es una política de eliminación. Es posible que los datos no se eliminen inmediatamente después del período TTL.
- Persistencia de secuencia. Elija guardar las secuencias en el sistema de archivos para conservar los datos en los reinicios del núcleo o guardar las secuencias en la memoria.
- Starting sequence number. Especifique el número de secuencia del mensaje que se utilizará como mensaje de inicio en la exportación.

Para obtener más información acerca de `MessageStreamDefinition`, consulte la referencia del SDK para el lenguaje de destino:

- MessageStreamDefinition en el SDK de Java
- MessageStreamDefinition en el SDK de Node.js
- MessageStreamDefinition en el SDK de Python

> **Note**
>
> `StreamManagerClient` también proporciona un destino que puede utilizar para exportar secuencias a un servidor HTTP. Este destino está pensado solo con fines de prueba. No es estable y no se admite para su uso en entornos de producción.

Una vez creado un flujo, las funciones de Lambda pueden anexar mensajes al flujo para enviar datos para su exportación y leer los mensajes del flujo para su procesamiento local. El número de secuencias que cree depende de sus capacidades de hardware y de su caso de negocio. Una estrategia es crear una secuencia para cada canal de destino en AWS IoT Analytics o secuencia de datos de Kinesis, aunque puede definir varios destinos para una secuencia. Una secuencia tiene una vida útil duradera.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.10.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

> **Note**
>
> La creación de flujos con un destino de exportación AWS IoT SiteWise o Amazon S3 tiene los siguientes requisitos:
>
> - Versión mínima AWS IoT Greengrass de Core: 1.11.0
> - Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Ejemplos

El siguiente fragmento crea una secuencia denominada `StreamName`. Define las propiedades de las secuencias en `MessageStreamDefinition` y los tipos de datos subordinados.

Python

```
client = StreamManagerClient()

try:
    client.create_message_stream(MessageStreamDefinition(
        name="StreamName",  # Required.
        max_size=268435456,  # Default is 256 MB.
        stream_segment_size=16777216,  # Default is 16 MB.
        time_to_live_millis=None,  # By default, no TTL is enabled.
        strategy_on_full=StrategyOnFull.OverwriteOldestData,  # Required.
        persistence=Persistence.File,  # Default is File.
        flush_on_write=False,  # Default is false.
        export_definition=ExportDefinition(  # Optional. Choose where/how the stream
 is exported to the Nube de AWS.
            kinesis=None,
            iot_analytics=None,
            iot_sitewise=None,
            s3_task_executor=None
        )
    ))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: create_message_stream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    client.createMessageStream(
            new MessageStreamDefinition()
                    .withName("StreamName") // Required.
                    .withMaxSize(268435456L)  // Default is 256 MB.
                    .withStreamSegmentSize(16777216L)  // Default is 16 MB.
                    .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
                    .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  //
 Required.
                    .withPersistence(Persistence.File)  // Default is File.
                    .withFlushOnWrite(false)  // Default is false.
```

```
                    .withExportDefinition(  // Optional. Choose where/how the stream
 is exported to the Nube de AWS.
                            new ExportDefinition()
                                    .withKinesis(null)
                                    .withIotAnalytics(null)
                                    .withIotSitewise(null)
                                    .withS3TaskExecutor(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del Java de Python: createMessageStream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.createMessageStream(
            new MessageStreamDefinition()
                .withName("StreamName") // Required.
                .withMaxSize(268435456)  // Default is 256 MB.
                .withStreamSegmentSize(16777216)  // Default is 16 MB.
                .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
                .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  //
 Required.
                .withPersistence(Persistence.File)  // Default is File.
                .withFlushOnWrite(false)  // Default is false.
                .withExportDefinition(  // Optional. Choose where/how the stream is
 exported to the Nube de AWS.
                    new ExportDefinition()
                        .withKinesis(null)
                        .withIotAnalytics(null)
                        .withIotSitewise(null)
                        .withS3TaskExecutor(null)
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
```

```
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del Node.js SDK: createMessageStream | MessageStreamDefinition

Para obtener más información acerca de la configuración de destinos de exportación, consulte the section called “Exportación de configuraciones para Nube de AWS destinos compatibles”.

Agregar un mensaje

Para enviar datos al administrador de flujos para su exportación, las funciones de Lambda anexan los datos al flujo de destino. El destino de exportación determina el tipo de datos que se van a transferir a este método.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.10.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

> **Note**
>
> La adición de mensajes a un destino de exportación AWS IoT SiteWise o Amazon S3 tiene los siguientes requisitos:
>
> - Versión mínima AWS IoT Greengrass de Core: 1.11.0
> - Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Ejemplos

AWS IoT Analytics o destinos de exportación de Kinesis Data Streams

El siguiente fragmento añade un mensaje a la secuencia denominada `StreamName`. Para los destinos de AWS IoT Analytics o Kinesis Data Streams, las funciones de Lambda anexan una masa de datos.

Este fragmento tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.10.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Python

```
client = StreamManagerClient()

try:
    sequence_number = client.append_message(stream_name="StreamName",
 data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: append_message

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
 array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del Java de Python: appendMessage

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const sequenceNumber = await client.appendMessage("StreamName",
 Buffer.from("Arbitrary byte array"));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del Node.js SDK: appendMessage

AWS IoT SiteWise destinos de exportación

El siguiente fragmento añade un mensaje a la secuencia denominada `StreamName`. Para destinos AWS IoT SiteWise, las funciones de Lambda anexan un objeto serializado `PutAssetPropertyValueEntry`. Para obtener más información, consulte the section called "Exportación a AWS IoT SiteWise".

> **Note**
>
> Cuando envía datos a AWS IoT SiteWise, los datos deben cumplir todos los requisitos de la acción `BatchPutAssetPropertyValue`. Para obtener más información, consulte BatchPutAssetPropertyValue en la Referencia de la API AWS IoT SiteWise.

Este fragmento tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.11.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Python

```
client = StreamManagerClient()
```

```
try:
    # SiteWise requires unique timestamps in all messages. Add some randomness to
 time and offset.

    # Note: To create a new asset property data, you should use the classes defined
 in the
    # greengrasssdk.stream_manager module.

    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
 offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
 timestamp=time_in_nanos)]
    putAssetPropertyValueEntry =
 PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
 property_alias="PropertyAlias", property_values=asset)
    sequence_number = client.append_message(stream_name="StreamName",
 data=Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: append_message | PutAssetPropertyValueEntry

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
 in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>() ;

    // IoTSiteWise requires unique timestamps in all messages. Add some randomness
 to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
```

```
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
            .withTimeInSeconds(Instant.now().getEpochSecond() -
 rand.nextInt(maxTimeRandomness))
            .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
 PutAssetPropertyValueEntry()
            .withEntryId(UUID.randomUUID().toString())
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues(entries);
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: appendMessage | PutAssetPropertyValueEntry

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const maxTimeRandomness = 60;
        const maxOffsetRandomness = 10000;
        const randomValue = Math.random();
        // Note: To create a new asset property data, you should use the classes
 defined in the
        // aws-greengrass-core-sdk StreamManager module.
        const timestamp = new TimeInNanos()
            .withTimeInSeconds(Math.round(Date.now() / 1000) -
 Math.floor(Math.random() - maxTimeRandomness))
            .withOffsetInNanos(Math.floor(Math.random() * maxOffsetRandomness));
        const entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);
```

```
        const putAssetPropertyValueEntry =  new PutAssetPropertyValueEntry()
            .withEntryId(`${ENTRY_ID_PREFIX}${i}`)
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues([entry]);
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: appendMessage | PutAssetPropertyValueEntry

Destinos de exportación de Amazon S3

El siguiente fragmento añade una tarea de exportación a la secuencia denominada `StreamName`. Para los destinos de Amazon S3, las funciones de Lambda añaden un objeto `S3ExportTaskDefinition` serializado que contiene información sobre el archivo de entrada de origen y el objeto de Amazon S3 de destino. Si el objeto especificado no existe, el administrador de flujos lo crea por usted. Para obtener más información, consulte the section called “Exportar a Amazon S3.”.

Este fragmento tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.11.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Python

```
client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
 bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
 data=Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
```

```
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: append_message | S3ExportTaskDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: appendMessage | S3ExportTaskDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
     // Append an Amazon S3 export task definition and print the sequence number.
     const taskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(taskDefinition)));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
```

```
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: appendMessage | S3ExportTaskDefinition

Lectura de mensajes

Leer mensajes de un flujo.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.10.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Ejemplos

El siguiente fragmento lee los mensajes de la secuencia denominada `StreamName`. El método read toma un objeto `ReadMessagesOptions` opcional que especifica el número de secuencia para comenzar a leer, los números mínimo y máximo para leer y un tiempo de espera para leer mensajes.

Python

```
client = StreamManagerClient()

try:
    message_list = client.read_messages(
        stream_name="StreamName",
        # By default, if no options are specified, it tries to read one message from
 the beginning of the stream.
        options=ReadMessagesOptions(
            desired_start_sequence_number=100,
            # Try to read from sequence number 100 or greater. By default, this is
 0.
            min_message_count=10,
            # Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is raised. By default, this is 1.
```

```
            max_message_count=100,  # Accept up to 100 messages. By default this is
 1.
            read_timeout_millis=5000
            # Try to wait at most 5 seconds for the min_messsage_count to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
        )
    )
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: read_messages, ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    List<Message> messages = client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                    // Try to read from sequence number 100 or greater. By default
 this is 0.
                    .withDesiredStartSequenceNumber(100L)
                    // Try to read 10 messages. If 10 messages are not available,
 then NotEnoughMessagesException is raised. By default, this is 1.
                    .withMinMessageCount(10L)
                    // Accept up to 100 messages. By default this is 1.
                    .withMaxMessageCount(100L)
                    // Try to wait at most 5 seconds for the min_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del Java de Python: readMessages | ReadMessagesOptions

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messages = await client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                // Try to read from sequence number 100 or greater. By default this
 is 0.
                .withDesiredStartSequenceNumber(100)
                // Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is thrown. By default, this is 1.
                .withMinMessageCount(10)
                // Accept up to 100 messages. By default this is 1.
                .withMaxMessageCount(100)
                // Try to wait at most 5 seconds for the minMessageCount to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                .withReadTimeoutMillis(5 * 1000)
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del Node.js SDK: readMessages | ReadMessagesOptions

Lista de secuencias

Obtenga la lista de flujos en el administrador de flujos.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.10.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Ejemplos

El siguiente fragmento de código obtiene una lista de las secuencias (por nombre) del administrador de secuencias.

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: list_streams

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del Java de Python: listStreams

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const streams = await client.listStreams();
    } catch (e) {
        // Properly handle errors.
```

```
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del Node.js SDK: listStreams

Descripción de una secuencia de mensajes

Obtenga metadatos sobre un flujo, incluida la definición, el tamaño y el estado de exportación del flujo.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.10.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Ejemplos

El siguiente fragmento de código obtiene metadatos de una secuencia llamada `StreamName`, como su definición, su tamaño y los estados del exportador.

Python

```
client = StreamManagerClient()

try:
    stream_description = client.describe_message_stream(stream_name="StreamName")
    if stream_description.export_statuses[0].error_message:
        # The last export of export destination 0 failed with some error
        # Here is the last sequence number that was successfully exported
        stream_description.export_statuses[0].last_exported_sequence_number

    if (stream_description.storage_status.newest_sequence_number >
            stream_description.export_statuses[0].last_exported_sequence_number):
```

```
        pass
        # The end of the stream is ahead of the last exported sequence number
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: describe_message_stream

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
 description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
        description.getExportStatuses().get(0).getLastExportedSequenceNumber();
    }

    if (description.getStorageStatus().getNewestSequenceNumber() >
            description.getExportStatuses().get(0).getLastExportedSequenceNumber())
 {
        // The end of the stream is ahead of the last exported sequence number.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del Java de Python: describeMessageStream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const description = await client.describeMessageStream("StreamName");
        const lastErrorMessage = description.exportStatuses[0].errorMessage;
        if (lastErrorMessage) {
            // The last export of export destination 0 failed with some error.
```

```
            // Here is the last sequence number that was successfully exported.
            description.exportStatuses[0].lastExportedSequenceNumber;
        }

        if (description.storageStatus.newestSequenceNumber >
            description.exportStatuses[0].lastExportedSequenceNumber) {
            // The end of the stream is ahead of the last exported sequence number.
        }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del Node.js SDK: describeMessageStream

Actualizar una secuencia de mensajes

Actualiza las propiedades de un flujo existente. Es posible que desee actualizar un flujo si sus requisitos cambian después de crearlo. Por ejemplo:

- Agregue una nueva configuración de exportación para un destino Nube de AWS.
- Aumente el tamaño máximo de un flujo para cambiar la forma en que se exportan o conservan los datos. Por ejemplo, el tamaño del flujo, en combinación con su estrategia en la configuración completa, puede provocar que los datos se eliminen o rechacen antes de que el administrador de flujos pueda procesarlos.
- Pausa y reanuda las exportaciones; por ejemplo, si las tareas de exportación son prolongadas y quiere racionar los datos de carga.

Las funciones de Lambda siguen este proceso de alto nivel para actualizar un flujo:

1. Consiga la descripción del flujo.
2. Actualice las propiedades de destino de los objetos correspondientes `MessageStreamDefinition` y subordinados.

3. Transfiera la actualización `MessageStreamDefinition`. Asegúrese de incluir las definiciones de objetos completas para el flujo actualizado. Las propiedades no definidas revierten a los valores predeterminados.

   Puede especificar el número de secuencia del mensaje que se utilizará como mensaje de inicio en la exportación.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.11.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Ejemplos

El siguiente fragmento de código actualiza la secuencia llamada `StreamName`. Actualiza varias propiedades de un flujo que se exporta a Kinesis Data Streams.

Python

```
client = StreamManagerClient()

try:
    message_stream_info = client.describe_message_stream(STREAM_NAME)
    message_stream_info.definition.max_size=536870912
    message_stream_info.definition.stream_segment_size=33554432
    message_stream_info.definition.time_to_live_millis=3600000
    message_stream_info.definition.strategy_on_full=StrategyOnFull.RejectNewData
    message_stream_info.definition.persistence=Persistence.Memory
    message_stream_info.definition.flush_on_write=False
    message_stream_info.definition.export_definition.kinesis=
        [KinesisConfig(
            # Updating Export definition to add a Kinesis Stream configuration.
            identifier=str(uuid.uuid4()), kinesis_stream_name=str(uuid.uuid4()))]
    client.update_message_stream(message_stream_info.definition)
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
```

```
    # Properly handle errors.
```

Referencia del SDK de Python: updateMessageStream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
 Strategy on full to reject new data.
           // Max Size update should be greater than initial Max Size defined in
 Create Message Stream request
            .withMaxSize(536870912L) // Update Max Size to 512 MB.
            .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
            .withFlushOnWrite(true) // Update flush on write to true.
            .withPersistence(Persistence.Memory) // Update the persistence to
 Memory.
            .withTimeToLiveMillis(3600000L)  // Update TTL to 1 hour.
            .withExportDefinition(
                // Optional. Choose where/how the stream is exported to the Nube de
 AWS.
                messageStreamInfo.getDefinition().getExportDefinition().
                    // Updating Export definition to add a Kinesis Stream
 configuration.
                    .withKinesis(new ArrayList<KinesisConfig>() {{
                        add(new KinesisConfig()
                            .withIdentifier(EXPORT_IDENTIFIER)
                            .withKinesisStreamName("test"));
                        }})
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: create_message_stream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
```

```
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
                // Max Size update should be greater than initial Max Size defined
 in Create Message Stream request
                .withMaxSize(536870912)  // Default is 256 MB. Updating Max Size to
 512 MB.
                .withStreamSegmentSize(33554432)  // Default is 16 MB. Updating
 Segment Size to 32 MB.
                .withTimeToLiveMillis(3600000)  // By default, no TTL is enabled.
 Update TTL to 1 hour.
                .withStrategyOnFull(StrategyOnFull.RejectNewData)  // Required.
 Updating Strategy on full to reject new data.
                .withPersistence(Persistence.Memory)  // Default is File. Update the
 persistence to Memory
                .withFlushOnWrite(true)  // Default is false. Updating to true.
                .withExportDefinition(
                    // Optional. Choose where/how the stream is exported to the Nube
 de AWS.
                    messageStreamInfo.definition.exportDefinition
                        // Updating Export definition to add a Kinesis Stream
 configuration.
                        .withKinesis([new
 KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: updateMessageStream | MessageStreamDefinition

Restricciones para actualizar los flujos

Se aplican las siguientes restricciones al actualizar flujos. A menos que se indique en la siguiente lista, las actualizaciones se aplican de inmediato.

- No puede actualizar la persistencia de un flujo. Para cambiar este comportamiento, elimine el flujo y crea un flujo que defina la nueva política de persistencia.
- Puede actualizar el tamaño máximo de una transmisión solo en las siguientes condiciones:
  - El tamaño máximo debe ser superior o igual al tamaño actual del flujo. Para encontrar esta información, describa la secuencia y, a continuación, compruebe el estado de almacenamiento del objeto `MessageStreamInfo` devuelto.
  - El tamaño máximo debe ser superior o igual al tamaño del segmento del flujo.
- Puede actualizar el tamaño del segmento de la transmisión a un valor inferior al tamaño máximo del flujo. La configuración actualizada se aplica a los segmentos nuevos.
- Las actualizaciones de la propiedad tiempo de vida (TTL) se aplican a las nuevas operaciones de anexión. Si reduce este valor, es posible que el administrador de flujos también elimine los segmentos existentes que superen el TTL.
- Las actualizaciones de la estrategia en toda la propiedad se aplican a las nuevas operaciones de anexión. Si establece la estrategia para sobrescribir los datos más antiguos, es posible que el administrados de flujos también sobrescriba los segmentos existentes en función del nuevo ajuste.
- Las actualizaciones de la propiedad de vaciar al escribir se aplican a los mensajes nuevos.
- Las actualizaciones de las configuraciones de exportación se aplican a las nuevas exportaciones. La solicitud de actualización debe incluir todas las configuraciones de exportación que desee admitir. De lo contrario, el administrador de flujos las elimina.
  - Al actualizar una configuración de exportación, especifique el identificador de la configuración de exportación de destino.
  - Para añadir una configuración de exportación, especifique un identificador único para la nueva configuración de exportación.
  - Para eliminar una configuración de exportación, omita la configuración de exportación.
- Para actualizar el número de secuencia inicial de una configuración de exportación en un flujo, debe especificar un valor inferior al último número de secuencia. Para encontrar esta información, describa la secuencia y, a continuación, compruebe el estado de almacenamiento del objeto `MessageStreamInfo` devuelto.

Eliminación de una secuencia de mensajes

Elimina un flujo. Cuando elimina una secuencia, todos los datos almacenados para la secuencia se eliminan del disco.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima AWS IoT Greengrass de Core: 1.10.0
- Versión mínima del SDK de AWS IoT Greengrass Core: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Ejemplos

El siguiente fragmento de código elimina la secuencia llamada `StreamName`.

Python

```
client = StreamManagerClient()

try:
    client.delete_message_stream(stream_name="StreamName")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: deleteMessageStream

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: delete_message_stream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.deleteMessageStream("StreamName");
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: deleteMessageStream

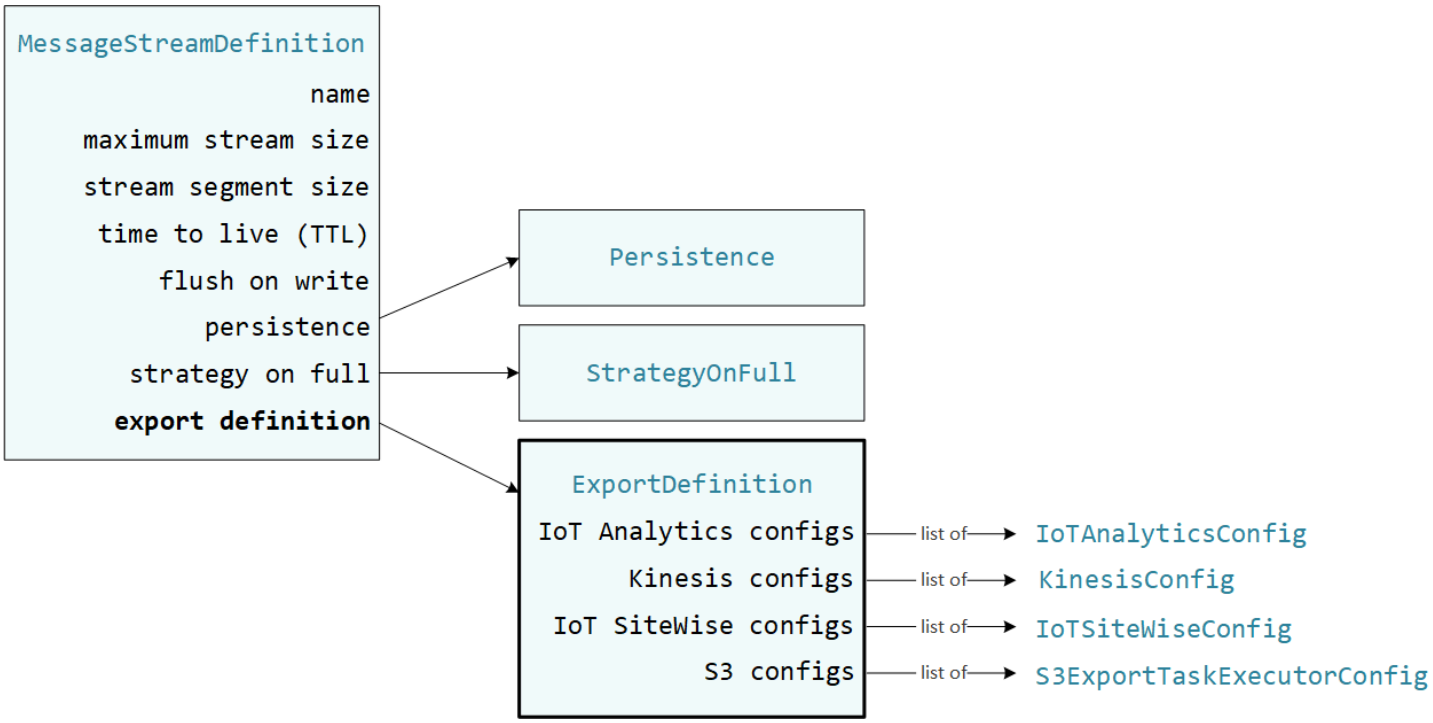
Véase también

- Administrar secuencias de datos
- the section called “Configurar el administrador de secuencias”
- the section called “Exportación de configuraciones para Nube de AWS destinos compatibles”
- the section called “Exportar flujos de datos (consola)”
- the section called “Exportar secuencias de datos (CLI)”
- `StreamManagerClient` en la referencia del SDK de AWS IoT Greengrass Core:
  - Python
  - Java
  - Node.js

Exportación de configuraciones para Nube de AWS destinos compatibles

Las funciones Lambda `StreamManagerClient`definidas de Lambda definidas por el usuario deben utilizar el Core SDK AWS IoT Greengrass para interactuar con el administrador de secuencias. Cuando una función de Lambda crea un flujo o actualiza un flujo, pasa un objeto `MessageStreamDefinition` que representa las propiedades del flujo, incluida la definición de exportación. El objeto `ExportDefinition` contiene las configuraciones de exportación definidas

para el flujo. El administrador de flujos utiliza estas configuraciones de exportación para determinar dónde y cómo exportar el flujo.



Puede definir cero o más configuraciones de exportación en un flujo, incluidas varias configuraciones de exportación para un único tipo de destino. Por ejemplo, puede exportar un flujo a dos canales AWS IoT Analytics y a un flujo de datos de Kinesis.

En caso de intentos fallidos de exportación, el administrador del flujo vuelve a intentar exportar los datos continuamente al Nube de AWS a intervalos de hasta cinco minutos. La cantidad de reintentos no tiene un límite máximo.

> **Note**
>
> `StreamManagerClient` también proporciona un destino que puede utilizar para exportar secuencias a un servidor HTTP. Este destino está pensado solo con fines de prueba. No es estable y no se admite para su uso en entornos de producción.

Destinos Nube de AWS admitidos

- Canales de AWS IoT Analytics
- Amazon Kinesis Data Streams
- AWS IoT SiteWise propiedades de activos

- Objetos de Amazon S3

Usted es responsable del mantenimiento de estos recursos Nube de AWS.

Canales de AWS IoT Analytics

El administrador de flujos admite la exportación automática a AWS IoT Analytics. AWS IoT Analytics le permite realizar análisis avanzados de sus datos para ayudarle a tomar decisiones de negocios y mejorar los modelos de machine learning. Para obtener más información, consulte ¿Qué es AWS IoT Analytics? en la AWS IoT Analytics Guía del usuario.

En el AWS IoT Greengrass Core SDK, las funciones de Lambda utilizan el `IoTAnalyticsConfig` para definir la configuración de exportación para este tipo de destino. Para obtener más información, consulte la referencia del SDK para el lenguaje de destino.

- IoTAnalyticsConfig en el SDK de Python
- IoTAnalyticsConfig en el SDK de Java
- IoTAnalyticsConfig en el SDK de Node.js

Requisitos

Este destino de exportación tiene los siguientes requisitos:

- Los canales de destino en AWS IoT Analytics deben estar en el mismo grupo Cuenta de AWS y Región de AWS en el grupo de Greengrass.
- El the section called “Rol de grupo de Greengrass” debe permitir el permiso `iotanalytics:BatchPutMessage` para segmentar los canales. Por ejemplo:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
```

```
            ]
        }
    ]
}
```

Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Exportación a AWS IoT Analytics

Para crear un flujo que se exporte a AWS IoT Analytics, las funciones de Lambda crean un flujo con una definición de exportación que incluye uno o más objetos `IoTAnalyticsConfig`. Este objeto define los ajustes de exportación, como el canal de destino, el tamaño del lote, el intervalo del lote y la prioridad.

Cuando sus funciones de Lambda reciben datos de los dispositivos, anexan mensajes que contienen una masa de datos al flujo de destino.

A continuación, el administrador de flujos exporta los datos en función de los ajustes del lote y la prioridad definidos en las configuraciones de exportación del flujo.

Amazon Kinesis Data Streams

El administrador de flujos permite exportar automáticamente a Amazon Kinesis Data Streams. Kinesis Data Streams se suele utilizar para agregar grandes volúmenes de datos y cargarlos en un almacenamiento de datos o en un clúster de map-reduce. Para obtener más información, consulte Qué son los Amazon Kinesis Data Streams en la Guía para desarrolladores de Amazon Kinesis.

En el AWS IoT Greengrass Core SDK, las funciones de Lambda utilizan el `KinesisConfig` para definir la configuración de exportación para este tipo de destino. Para obtener más información, consulte la referencia del SDK para el lenguaje de destino.

- KinesisConfig en el SDK de Python
- KinesisConfig en el SDK de Java
- KinesisConfig en el SDK de Node.js

Requisitos

Este destino de exportación tiene los siguientes requisitos:

- Las flujos de destino de Kinesis Data Streams deben estar en el mismo Cuenta de AWS y Región de AWS como grupo de Greengrass.
- El the section called “Rol de grupo de Greengrass” debe permitir el permiso `kinesis:PutRecords` para destinarse a flujos de datos. Por ejemplo:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/stream_1_name",
                "arn:aws:kinesis:region:account-id:stream/stream_2_name"
            ]
        }
    ]
}
```

  Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Exportar a Kinesis Data Streams

Para crear un flujo que se exporte a Kinesis Data Streams, sus funciones de Lambda crean un flujo con una definición de exportación que incluye uno o más objetos `KinesisConfig`. Este objeto define los ajustes de exportación, como el flujo de datos de destino, el tamaño del lote, el intervalo del lote y la prioridad.

Cuando sus funciones de Lambda reciben datos de los dispositivos, anexan mensajes que contienen una masa de datos al flujo de destino. A continuación, el administrador de flujos exporta los datos en función de los ajustes del lote y la prioridad definidos en las configuraciones de exportación del flujo.

El administrador de flujos genera un UUID aleatorio único como clave de partición para cada registro cargado en Amazon Kinesis.

AWS IoT SiteWise propiedades de activos

El administrador de flujos admite la exportación automática a AWS IoT SiteWise. AWS IoT SiteWise le permite recopilar, organizar y analizar datos de equipos industriales a escala. Para obtener más información, consulte ¿Qué es AWS IoT SiteWise? en la AWS IoT SiteWiseGuía del usuario.

En el AWS IoT Greengrass Core SDK, las funciones de Lambda utilizan el `IoTSiteWiseConfig` para definir la configuración de exportación para este tipo de destino. Para obtener más información, consulte la referencia del SDK para el lenguaje de destino.

- IoTSiteWiseConfig en el SDK de Python
- IoTSiteWiseConfig en el SDK de Java
- IoTSiteWiseConfig en el SDK de Node.js

> **Note**
>
> AWS también proporciona la the section called “IoT SiteWise”, que es una solución prediseñada que puede utilizar con fuentes OPC-UA.

Requisitos

Este destino de exportación tiene los siguientes requisitos:

- Las propiedades de los activos de destino en AWS IoT SiteWise deben estar en el mismo Cuenta de AWS y Región de AWS que el grupo Greengrass.

  > **Note**
  >
  > Para la lista de las regiones que AWS IoT SiteWise admite, consulte AWS IoT SiteWiseCuotas y puntos de enlace en la Referencia general de AWS.

- the section called “Rol de grupo de Greengrass”El `iotsitewise:BatchPutAssetPropertyValue` debe permitir que el permiso se

dirija a las propiedades de los activos. El siguiente ejemplo de política utiliza la clave `iotsitewise:assetHierarchyPath` de condición para conceder acceso a un activo raíz de destino y a sus elementos secundarios. Puede quitar `Condition` de la política para permitir el acceso a todos sus recursos AWS IoT SiteWise o especificar ARN de los activos individuales.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Para obtener información de seguridad importante, consulte la autorización BatchPutAssetPropertyValue en la AWS IoT SiteWiseGuía del usuario.

Exportación a AWS IoT SiteWise

Para crear un flujo que se exporte a AWS IoT SiteWise, las funciones de Lambda crean un flujo con una definición de exportación que incluye uno o más objetos `IoTSiteWiseConfig`. Este objeto define los ajustes de exportación, como el tamaño del lote, el intervalo del lote y la prioridad.

Cuando sus funciones de Lambda reciben datos de propiedad de activos, anexan mensajes que contienen los datos al flujo de destino. Los mensajes son objetos `PutAssetPropertyValueEntry` serializados en JSON que contienen los valores de propiedad de una o más propiedades de

los activos. Para obtener más información, consulte Añadir un mensaje para los AWS IoT SiteWisedestinos de exportación.

> **Note**
>
> Cuando envía datos a AWS IoT SiteWise, los datos deben cumplir todos los requisitos de la acción `BatchPutAssetPropertyValue`. Para obtener más información, consulte BatchPutAssetPropertyValue en la Referencia de la API AWS IoT SiteWise.

A continuación, el administrador de flujos exporta los datos en función de los ajustes del lote y la prioridad definidos en las configuraciones de exportación del flujo.

Puede ajustar los ajustes del administrador de flujos y la lógica de la función de Lambda para diseñar su estrategia de exportación. Por ejemplo:

- Para realizar exportaciones prácticamente en tiempo real, establezca ajustes del tamaño de lote e intervalos bajos y añada los datos al flujo cuando lo reciba.
- Para optimizar el procesamiento por lotes, mitigar las restricciones de ancho de banda o minimizar los costos, las funciones de Lambda pueden agrupar los puntos de datos de marca temporal, calidad y valor (timestamp-quality-value, TQV) recibidos para una sola propiedad de activo antes de agregar los datos al flujo. Una estrategia consiste en agrupar las entradas de hasta 10 combinaciones diferentes de propiedades y activos (o alias de propiedades) en un mensaje, en lugar de enviar más de una entrada para la misma propiedad. Esto ayuda al Administrador de flujos a mantenerse dentro de las cuotas de AWS IoT SiteWise.

Objetos de Amazon S3

El administrador de flujos permite exportar automáticamente a Amazon S3. Puede utilizar Amazon S3 para almacenar y recuperar grandes cantidades de datos. Para obtener más información, consulte ¿Qué es Amazon S3? en la Guía para desarrolladores de Amazon Simple Storage Service.

En el AWS IoT Greengrass Core SDK, las funciones de Lambda utilizan el `S3ExportTaskExecutorConfig` para definir la configuración de exportación para este tipo de destino. Para obtener más información, consulte la referencia del SDK para el lenguaje de destino.

- S3ExportTaskExecutorConfig en el SDK de Python
- S3ExportTaskExecutorConfig en el SDK de Java
- S3ExportTaskExecutorConfig en el SDK de Node.js

Requisitos

Este destino de exportación tiene los siguientes requisitos:

- Los buckets de Amazon S3 de destino deben estar en la misma Cuenta de AWS que el grupo de Greengrass.
- Si la contenerización predeterminada para el grupo de Greengrass es el contenedor de Greengrass, debe configurar el parámetro STREAM_MANAGER_READ_ONLY_DIRS para usar un directorio de archivos de entrada que esté en `/tmp` o no esté en el sistema de archivos raíz.
- Si una función de Lambda que se ejecuta en el modo contenedor de Greengrass escribe archivos de entrada en el directorio de archivos de entrada, usted debe crear un recurso de volumen local para el directorio y montar el directorio en el contenedor con permisos de escritura. Esto garantiza que los archivos se escriban en el sistema de archivos raíz y que sean visibles fuera del contenedor. Para obtener más información, consulte Acceder a recursos locales.
- El the section called “Rol de grupo de Greengrass” debe permitir los siguientes permisos para los buckets de destino. Por ejemplo:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-1-name/*",
                "arn:aws:s3:::bucket-2-name/*"
            ]
        }
    ]
}
```

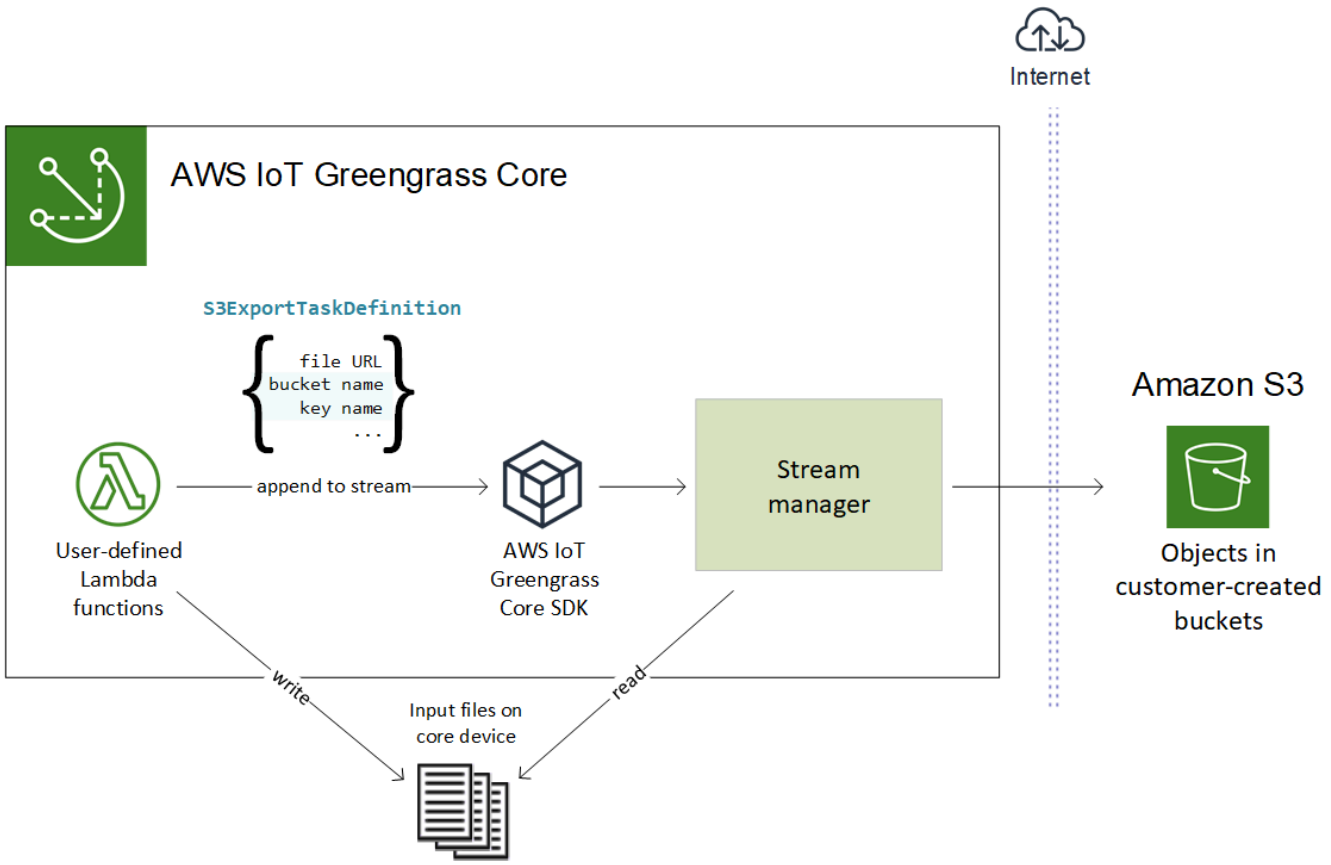
Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Exportar a Amazon S3.

Para crear un flujo que se exporte a Amazon S3, las funciones de Lambda utilizan el objeto `S3ExportTaskExecutorConfig` para configurar la política de exportación. La política define los ajustes de exportación, como el umbral de carga multiparte y la prioridad. Para las exportaciones de Amazon S3, el administrador de flujos carga los datos que lee de los archivos locales en el dispositivo principal. Para iniciar una carga, sus funciones de Lambda anexan una tarea de exportación al flujo de destino. La tarea de exportación contiene información sobre el archivo de entrada y el objeto Amazon S3 de destino. El administrador de flujos ejecuta las tareas en la secuencia en que se anexan al flujo.

> **Note**
>
> El bucket de destino ya debe existir en su Cuenta de AWS. Si no existe un objeto para la clave especificada, el administrador de flujos crea el objeto automáticamente.

Este flujo de alto nivel se muestra en el siguiente diagrama.



El administrador de flujos utiliza la propiedad de umbral de carga multiparte, el ajuste del tamaño mínimo de las piezas y el tamaño del archivo de entrada para determinar cómo cargar los datos. El umbral de carga multiparte debe ser igual o mayor que el tamaño mínimo de la pieza. Si desea cargar datos en paralelo, puede crear varios flujos.

Las claves que especifican los objetos de Amazon S3 de destino pueden incluir cadenas Java DateTimeFormatter válidas en los marcadores de posición `!{timestamp:value}`. Puede utilizar estos marcadores de fecha y hora para particionar los datos en Amazon S3 en función de la hora en la que se cargaron los datos del archivo de entrada. Por ejemplo, el siguiente nombre de clave se resuelve en un valor como `my-key/2020/12/31/data.txt`.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> Si desea supervisar el estado de exportación de un flujo, cree primero un flujo de estado y, a continuación, configure el flujo de exportación para utilizarlo. Para obtener más información, consulte the section called “Supervise las tareas de exportación”.

Administrar datos de entrada

Puede crear un código que las aplicaciones de IoT usen para administrar el ciclo de vida de los datos de entrada. El siguiente ejemplo de flujo de trabajo muestra cómo se pueden utilizar las funciones de Lambda para administrar estos datos.

1. Un proceso local recibe datos de dispositivos o periféricos y, a continuación, los escribe en los archivos de un directorio del dispositivo principal. Estos son los archivos de entrada del administrador de flujos.

   > **Note**
   >
   > Para determinar si debe configurar el acceso al directorio de archivos de entrada, consulte el parámetro STREAM_MANAGER_READ_ONLY_DIRS.
   > El proceso en el que se ejecuta el administrador de flujos hereda todos los permisos del sistema de archivos de la identidad de acceso predeterminada del grupo. El administrador de flujos debe tener permiso de acceso a los archivos de entrada. Puede utilizar el comando `chmod(1)` para cambiar el permiso de los archivos, si es necesario.

2. Una función de Lambda escanea el directorio y anexa una tarea de exportación a la secuencia de destino cuando se crea un archivo nuevo. La tarea es un objeto `S3ExportTaskDefinition` serializado en JSON que especifica la URL del archivo de entrada, el bucket y la clave de Amazon S3 de destino y los metadatos de usuario opcionales.
3. El administrador de flujos lee el archivo de entrada y exporta los datos a Amazon S3 en el orden de las tareas anexas. El bucket de destino ya debe existir en su Cuenta de AWS. Si no existe un objeto para la clave especificada, el administrador de flujos crea el objeto automáticamente.
4. La función de Lambda lee los mensajes de un flujo de estado para supervisar el estado de la exportación. Una vez finalizadas las tareas de exportación, la función de Lambda puede eliminar los archivos de entrada correspondientes. Para obtener más información, consulte the section called “Supervise las tareas de exportación”.

Supervise las tareas de exportación

Puede crear un código que las aplicaciones de IoT utilizan para monitorear el estado de sus exportaciones de Amazon S3. Las funciones de Lambda deben crear un flujo de estado y, a continuación, configurar el flujo de exportación para escribir actualizaciones de estado en el flujo de estado. Una sola transmisión de estado puede recibir actualizaciones de estado de varias transmisiones que se exportan a Amazon S3.

En primer lugar, cree un flujo para utilizarlo como flujo de estado. Puede configurar las políticas de tamaño y retención del flujo para controlar la vida útil de los mensajes de estado. Por ejemplo:

- Configure `Persistence` en `Memory` si no desea guardar los mensajes de estado.
- Configure `StrategyOnFull` en `OverwriteOldestData` para que no se pierdan los nuevos mensajes de estado.

A continuación, cree o actualice el flujo de exportación para usar el flujo de estado.
En concreto, defina la propiedad de configuración de estado de la configuración de `S3ExportTaskExecutorConfig` exportación del flujo. Esto le indica al administrador del flujo que escriba mensajes de estado sobre las tareas de exportación en el flujo de estado. En el objeto `StatusConfig`, especifique el nombre del flujo de estado y el nivel de detalle. Los siguientes valores admitidos van desde el menos detallado (`ERROR`) al más detallado (). `TRACE` El valor predeterminado es `INFO`.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`

El siguiente ejemplo de flujo de trabajo muestra cómo las funciones de Lambda pueden utilizar un flujo de estado para supervisar el estado de la exportación.

1. Como se describió en el flujo de trabajo anterior, una función de Lambda anexa una tarea de exportación a un flujo que está configurado para escribir mensajes de estado sobre las tareas

de exportación en un flujo de estado. La operación de incorporación devuelve un número de secuencia que representa el ID de la tarea.

2. Una función de Lambda lee los mensajes secuencialmente del flujo de estado y, a continuación, filtra los mensajes en función del nombre del flujo y el identificador de la tarea o en función de una propiedad de la tarea de exportación del contexto del mensaje. Por ejemplo, la función de Lambda puede filtrar por la URL del archivo de entrada de la tarea de exportación, que está representada por el objeto `S3ExportTaskDefinition` en el contexto del mensaje.

   Los siguientes códigos de estado indican que una tarea de exportación ha alcanzado un estado completo:

   - `Success`. Se ha completado correctamente la carga.
   - `Failure`. El administrador de flujos detectó un error; por ejemplo, el bucket especificado no existe. Tras resolver el problema, puede volver a añadir la tarea de exportación al flujo.
   - `Canceled`. La tarea se canceló porque se eliminó la definición de transmisión o exportación o porque venció el período de vida (TTL) de la tarea.

   > **Note**
   >
   > La tarea también puede tener un estado de `InProgress` o `Warning`. El administrador de flujos emite advertencias cuando un evento devuelve un error que no afecta a la ejecución de la tarea. Por ejemplo, si no se limpia una carga parcial cancelada, aparecerá una advertencia.

3. Una vez finalizadas las tareas de exportación, la función de Lambda puede eliminar los archivos de entrada correspondientes.

El siguiente ejemplo muestra cómo una función de Lambda puede leer y procesar los mensajes de estado.

Python

```
import time
from greengrasssdk.stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
```

```
    StreamManagerClient,
)
from greengrasssdk.stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
 read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
 StatusMessage)

                # Check the status of the status message. If the status is
 "Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
 unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
 status message.
                # If the status was "InProgress", the status indicates that the
 server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
 to S3.")
                    is_file_uploaded_to_s3 = True
                elif status_message.status == Status.Failure or
 status_message.status == Status.Canceled:
                    logger.info(
                        "Unable to upload file at path " + file_url + " to S3.
 Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
            time.sleep(5)
        except StreamManagerException:
            logger.exception("Exception while running")
```

```
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: read_messages | StatusMessage

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
 ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
                    StatusMessage statusMessage =
 ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
 StatusMessage.class);
                    // Check the status of the status message. If the status is
 "Success", the file was successfully uploaded to S3.
                    // If the status was either "Failure" or "Canceled", the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
```

```
                        System.out.println("Successfully uploaded file at path " +
 FILE_URL + " to S3.");
                        isS3UploadComplete = true;
                    } else if (Status.Failure.equals(statusMessage.getStatus()) ||
 Status.Canceled.equals(statusMessage.getStatus())) {
                        System.out.println(String.format("Unable to upload file at
 path %s to S3. Message %s",

 statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
                            statusMessage.getMessage()));
                        sS3UploadComplete = true;
                    }
                }
            } catch (StreamManagerException ignored) {
            } finally {
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                Thread.sleep(5000);
            }
        } catch (e) {
        // Properly handle errors.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: readMessages | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require('aws-greengrass-core-sdk').StreamManager;

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
```

```
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));

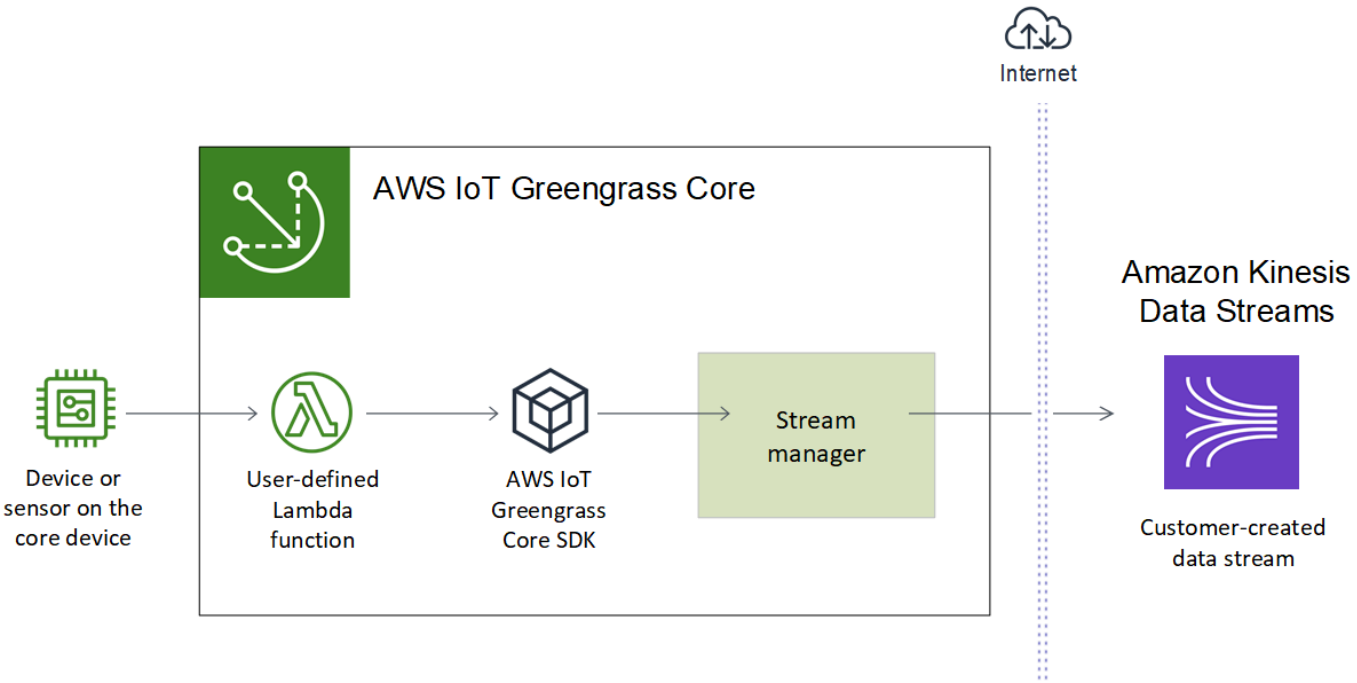
                messages.forEach((message) => {
                    // Deserialize the status message first.
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to S3.
                    // If the status was either 'Failure' or 'Cancelled', the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (statusMessage.status === Status.Success) {
                        console.log(`Successfully uploaded file at path ${FILE_URL}
 to S3.`);
                        isS3UploadComplete = true;
                    } else if (statusMessage.status === Status.Failure ||
 statusMessage.status === Status.Canceled) {
                        console.log(`Unable to upload file at path ${FILE_URL} to
 S3. Message: ${statusMessage.message}`);
                        isS3UploadComplete = true;
                    }
                });
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                await new Promise((r) => setTimeout(r, 5000));
            } catch (e) {
                // Ignored
            }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: readMessages | StatusMessage

Exportar flujos de datos a la Nube de AWS (consola)

En este tutorial se muestra cómo utilizar la consola de AWS IoT para configurar e implementar un grupo de AWS IoT Greengrass con el gestor de secuencias habilitado. El grupo contiene una función de Lambda definida por el usuario que escribe en una secuencia en el administrador de secuencias, que luego se exporta automáticamente a la Nube de AWS.

El administrador de secuencias hace que la asimilación, el procesamiento y la exportación de flujos de datos de gran volumen sea más eficiente y fiable. En este tutorial, va a crear una función de Lambda de `TransferStream` que consume datos de IoT. La función de Lambda utiliza el SDK de AWS IoT Greengrass Core para crear una secuencia en el administrador de secuencias y luego leer y escribir en ella. El administrador de secuencias exporta la secuencia al Flujo de datos Kinesis. En el siguiente diagrama se muestra este flujo de trabajo.



El objetivo de este tutorial es mostrar cómo utilizan las funciones de Lambda definidas por el usuario el objeto de `StreamManagerClient` en el SDK de AWS IoT Greengrass Core para interactuar con el administrador de flujos. Para simplificar, la función de Lambda que va a crear en este tutorial genera datos de dispositivos simulados.

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Un grupo de Greengrass y un núcleo de Greengrass (versión 1.10 o posterior). Para obtener información acerca de cómo crear un núcleo y un grupo de Greengrass, consulte Empezando con AWS IoT Greengrass. El tutorial de introducción también incluye pasos para instalar el software AWS IoT Greengrass Core.

  > **Note**
  >
  > El administrador de secuencias no es compatible con las distribuciones OpenWrt.

- Java 8 Runtime (JDK 8) instalado en el dispositivo principal.
  - Para distribuciones basadas en Debian (incluido Raspbian) o distribuciones basadas en Ubuntu, ejecute el siguiente comando:

    ```
    sudo apt install openjdk-8-jdk
    ```

  - Para distribuciones basadas en Red Hat (incluido Amazon Linux), ejecute el siguiente comando:

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Para obtener más información, consulte How to download and install prebuilt OpenJDK packages (Cómo descargar e instalar paquetes OpenJDK preconfigurados) en la documentación de OpenJDK.

- SDK de AWS IoT Greengrass Core para Python versión 1.5.0 o posteriores Para usar `StreamManagerClient` en el SDK de AWS IoT Greengrass Core para Python, debe:
  - Instalar Python 3.7 o versiones posteriores en el dispositivo principal.
  - Incluya el SDK y sus dependencias en su paquete de implementación de la función de Lambda. Se incluyen las instrucciones en este tutorial.

  > **Tip**
  >
  > Puede usar `StreamManagerClient` con Java o NodeJS. Para ver código de ejemplo, consulte el SDK de AWS IoT Greengrass Core para Java y el SDK de AWS IoT Greengrass Core para Node.js en GitHub.

- Una secuencia de destino denominada **`MyKinesisStream`** creada en Amazon Kinesis Data Streams en la misma Región de AWS que su grupo de Greengrass. Para obtener más información, consulte Crear un flujo en la Guía para desarrolladores de Amazon Kinesis.

> **Note**
>
> En este tutorial, el administrador de secuencias exporta datos Kinesis Data Streams, lo que deriva en cargos a su Cuenta de AWS. Para obtener información acerca de los precios, consulte Precios de Kinesis Data Streams.
> Para no incurrir en gastos, puede ejecutar este tutorial sin crear una secuencia de datos Kinesis. En este caso, compruebe los registros para ver si el administrador de flujos intentó exportar la secuencia al flujo de datos Kinesis.

- Una política de IAM agregada al the section called “Rol de grupo de Greengrass” que permita la acción de `kinesis:PutRecords` en el flujo de datos de destino, tal y como se muestra en el siguiente ejemplo:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

El tutorial contiene los siguientes pasos generales:

1. Creación de un paquete de implementación de la función de Lambda
2. Creación de una función de Lambda
3. Agregar una función al grupo
4. Habilitar el administrador de secuencias
5. Configurar el registro local
6. Implementar el grupo
7. Pruebe la aplicación.

Completar el tutorial debería tomarle aproximadamente 20 minutos.

Paso 1: Crear un paquete de implementación de la función de Lambda

En este paso, va a crear un paquete de implementación de funciones de Lambda que contiene código de función y dependencias de Python. Cargará este paquete más adelante cuando cree la función de Lambda en AWS Lambda. La función de Lambda utiliza el SDK de AWS IoT Greengrass Core para crear secuencias locales e interactuar con ellas.

> **Note**
>
> Sus funciones de Lambda definidas por el usuario deben utilizar el SDK de AWS IoT Greengrass Core para interactuar con el administrador de flujo. Para obtener más información sobre los requisitos del administrador de secuencias de Greengrass, consulte Requisitos del administrador de secuencias de Greengrass.

1. Descargue el SDK de AWS IoT Greengrass Core para Python versión 1.5.0 o posteriores.
2. Descomprima el paquete descargado para obtener el SDK. El SDK es la carpeta `greengrasssdk`.
3. Instale dependencias de paquetes para incluirlas con el SDK en su paquete de implementación de funciones de Lambda.

   1. Vaya al directorio de SDK que contiene el archivo de `requirements.txt`. Este archivo registra las dependencias.
   2. Instale las dependencias del SDK. Por ejemplo, ejecute el siguiente comando de `pip` para instalarlas en el directorio actual:

      ```
      pip install --target . -r requirements.txt
      ```

4. Guarde la siguiente función de código de Python en un archivo local llamado "`transfer_stream.py`".

   > **Tip**
   >
   > Para ver un ejemplo de código que utiliza Java y NodeJS, consulte el SDK de AWS IoT Greengrass Core para Java y SDK de AWS IoT Greengrass Core para Node.js en GitHub.

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
 exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
```

```
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
```

```
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. Comprima en un archivo ZIP los siguientes elementos en un archivo denominado "`transfer_stream_python.zip`". Este es el paquete de implementación de la función de Lambda.

   - transfer_stream.py. Lógica de la aplicación.
   - greengrasssdk. Biblioteca necesaria para las funciones de Lambda de Python Greengrass que publican mensajes MQTT.

     Las Operaciones del administrador de secuencias están disponibles en la versión 1.5.0 o en las versiones posteriores del SDK de AWS IoT Greengrass Core para Python.
   - Las dependencias que instaló para el SDK de AWS IoT Greengrass Core para Python (por ejemplo, los directorios de `cbor2`).

   Al crear el archivo de `zip`, incluya solo estos elementos, no la carpeta que los contiene.

Paso 2: crear una función Lambda

En este paso, va a utilizar la consola de AWS Lambda para crear una función de Lambda y va a configurarla para utilizar su paquete de implementación. A continuación, publicará una versión de la función y creará un alias.

1. Primero, cree una función de Lambda.

   a. En la AWS Management Console, elija Services (Servicios) y abra la consola de AWS Lambda.

   b. Elija Crear función, y, a continuación, Autor desde cero.

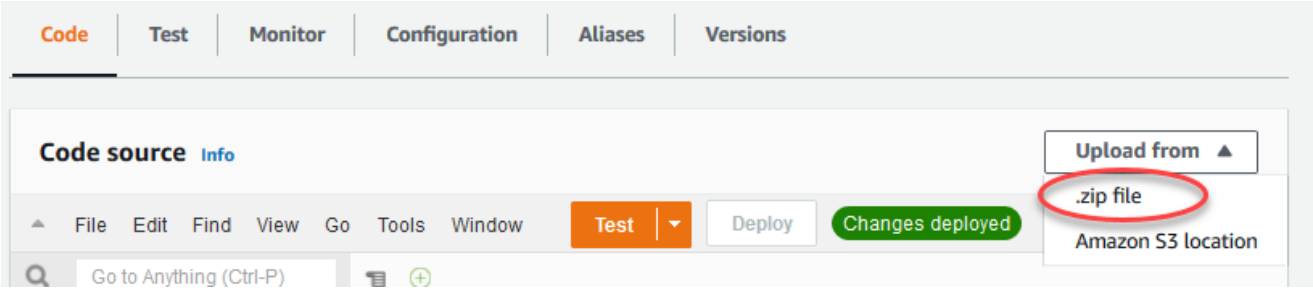
c. En la sección Basic information (Información básica), utilice los siguientes valores:

- En Function name (Nombre de la función), introduzca **`TransferStream`**.
- En Runtime (Tiempo de ejecución), elija Python 3.7.
- En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol.

d. En la parte inferior de la página, elija Create function.

2. A continuación, registre el controlador y cargue el paquete de implementación de la función de Lambda.

a. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip.



b. Seleccione Cargar y, a continuación, elija su paquete de implementación `transfer_stream_python.zip`. A continuación, elija Save (Guardar).

c. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

- En Runtime (Tiempo de ejecución), elija Python 3.7.
- En Handler (Controlador), escriba **`transfer_stream.function_handler`**.

d. Seleccione Save.

> **Note**
>
> El botón de prueba de la consola de AWS Lambda no funciona con esta función. El SDK AWS IoT Greengrass Core no contiene los módulos necesarios para ejecutar las funciones de Lambda de Greengrass de forma independiente en la consola AWS Lambda. Estos módulos (por ejemplo, `greengrass_common`) se suministran a las funciones una vez desplegados en el núcleo de Greengrass.

3. Ahora, publique la primera versión de su función de Lambda y cree un alias para la versión.

> **Note**
>
> Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

a. En el menú Actions, elija Publish new version.
b. En Version description (Descripción de versión), escriba **`First version`** y, a continuación, elija Publish (Publicar).
c. En la página de configuración de TransferStream: 1, en el menú Actions (Acciones), elija Create alias (Crear alias).
d. En la página Create a new alias, utilice los valores siguientes:

   - En Name (Nombre), ingrese **`GG_TransferStream`**.
   - En Version (Versión), elija 1.

   > **Note**
   >
   > AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST.

e. Seleccione Create (Crear).

Ahora está preparado para añadir la función de Lambda al grupo de Greengrass.

Paso 3: Agregar una función de Lambda al grupo de Greengrass

En este paso, va a añadir la función de Lambda al grupo y a configurar el ciclo de vida y las variables de entorno. Para obtener más información, consulte the section called “Control de la ejecución de la función de Lambda de Greengrass”.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.

3. En la página de configuración del grupo, elija la pestaña Funciones de lambda.
4. En la sección Mis funciones de Lambda, seleccione Añadir.
5. En la página Agregar función de Lambda, seleccione función de Lambda para su función de lambda.
6. Para la versión de Lambda, seleccione Alias:GG_TransferStream.

   Ahora, configure las propiedades que determinan el comportamiento de la función de Lambda en el grupo Greengrass.
7. En la sección de configuración de la función de Lambda, realice los siguientes cambios:

   - Establezca el Memory limit (Límite de memoria) en 32 MB.
   - En Ancladas, elija Verdadero

   > **Note**
   >
   > Una función de Lambda de larga duración (o anclada) se inicia automáticamente después de arrancar AWS IoT Greengrass y sigue ejecutándose en su propio contenedor. Esto contrasta con una función de Lambda bajo demanda, que se inicia cuando se la invoca y se detiene cuando no quedan tareas que ejecutar. Para obtener más información, consulte the section called “Configuración del ciclo de vida”.
8. Elija Añadir función de Lambda.

Paso 4: Habilitar el administrador de secuencias

En este paso, asegúrese de que el administrador de secuencias está habilitado.

1. En la página de configuración del grupo, elija la pestaña Funciones de lambda.
2. En Funciones de Lambda del sistema, seleccione el administrador de secuencias y luego verifique el estado. Si está deshabilitado, elija Edit (Editar). A continuación, elija Enable (Activar) y Save (Guardar). Puede utilizar la configuración de parámetros predeterminada para este tutorial. Para obtener más información, consulte the section called “Configurar el administrador de secuencias”.

> **Note**
>
> Cuando utiliza la consola para habilitar el administrador de secuencias y desplegar el grupo, el tamaño de la memoria para el administrador de transmisiones se establece por defecto en 4194304 KB (4 GB). Se recomienda establecer el tamaño de la memoria en al menos 128000 KB.

Paso 5: Configurar el registro local

En este paso, debe configurar los componentes del sistema de AWS IoT Greengrass, las funciones de Lambda definidas por el usuario y los conectores en el grupo para que escriban los registros en el sistema de archivos del dispositivo del núcleo. Puede usar registros para solucionar cualquier problema que pueda surgir. Para obtener más información, consulte the section called “Monitorización con registros de AWS IoT Greengrass”.

1. En Local logs configuration (Configuración de registros locales), compruebe si el registro local está configurado.
2. Si los registros no están configurados para componentes del sistema Greengrass o para funciones de Lambda definidas por el usuario, seleccione Editar.
3. Elija el nivel de registro de funciones de Lambda del usuario y el nivel de registro del sistema Greengrass.
4. Conserve los valores predeterminados para el nivel de registro y el límite de espacio en disco y, a continuación, elija Guardar.

Paso 6: Implementar el grupo de Greengrass

Implemente el grupo en el dispositivo del núcleo.

1. Asegúrese de que el núcleo de AWS IoT Greengrass se está ejecutando. Ejecute los siguientes comandos en el terminal de Raspberry Pi según sea necesario.

   a. Para comprobar si el demonio está en ejecución:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, el demonio está en ejecución.

> **Note**
>
> La versión que figura en la ruta depende de la versión del software AWS IoT Greengrass Core que esté instalada en el dispositivo del núcleo.

b. Iniciar el daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. En la página de configuración de grupo, elija Implementar.
3. a. En la pestaña Funciones de Lambda, en la sección Funciones de Lambda del sistema, seleccione Detector IP y elija Editar.

   b. En el cuadro de diálogo Editar configuración del detector IP, seleccione Detectar y anular automáticamente los puntos de conexión del agente MQTT.

   c. Seleccione Save.

   Esto permite a los dispositivos adquirir automáticamente la información de conexión del dispositivo principal, como la dirección IP, el DNS y el número de puerto. Se recomienda la detección automática, pero AWS IoT Greengrass también es compatible con puntos de enlace especificados manualmente. Solo se le solicitará el método de detección la primera vez que se implemente el grupo.

   > **Note**
   >
   > Si se le solicita, conceda permiso para crear el rol de servicio de Greengrass y asócielo a su Cuenta de AWS en el Región de AWS actual. Este rol permite a AWS IoT Greengrass acceder a los servicios de AWS.

   En la página Deployments (Implementaciones), se muestra la marca temporal, el ID de versión y el estado de la implementación. Una vez terminada, la implementación debería mostrar el estado Completado.

Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

Paso 7: Probar la aplicación

La función de Lambda `TransferStream` genera datos simulados del dispositivo. Escribe datos en una secuencia que el administrador de secuencias exporta a la secuencia de datos de Kinesis de destino.

1. En la consola de Amazon Kinesis , bajo las Secuencias de datos de Kinesis, seleccione MyKinesisStream.

   > **Note**
   >
   > Si ejecutó el tutorial sin una secuencia de datos de Kinesis de destino, compruebe el archivo de registro del administrador de secuencias (`GGStreamManager`). Si contiene `export stream MyKinesisStream doesn't exist` en un mensaje de error, la prueba se ha realizado correctamente. Este error significa que el servicio intentó exportar a la secuencia, pero que la secuencia no existe.

2. En la página MyKinesisStream, seleccione Monitoring (Supervisión). Si la prueba se realiza correctamente, debería ver los datos en los gráficos Put Records. En función de la conexión, es posible que tarde un minuto en mostrar los datos.

   > **Important**
   >
   > Cuando haya terminado la prueba, elimine la secuencia de datos de Kinesis para evitar incurrir en más gastos.
   > O ejecute el siguiente comando para detener el daemon de Greengrass. Así evitará que el núcleo envíe mensajes hasta que esté listo para continuar las pruebas.
   >
   > ```
   > cd /greengrass/ggc/core/
   > sudo ./greengrassd stop
   > ```

3. Elimine la función de Lambda TransferStream del núcleo.

   a. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).

b. En Grupos de Greengrass, elija su grupo.

c. En la página Lambdas seleccione los puntos suspensivos (...) para la función de TransferStream y, a continuación, seleccione Remove function (Eliminar función).

d. En Actions (Acciones), seleccione Deploy (Implementar).

Para ver la información de registro o solucionar problemas con las secuencias, compruebe los registros para las funciones `TransferStream` y `GGStreamManager`. Debe tener permisos de `root` para leer registros AWS IoT Greengrass en el sistema de archivos.

- `TransferStream` escribe entradas de registro en *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/`*`account-id`*`/TransferStream.log`.
- `GGStreamManager` escribe entradas de registro en *`greengrass-root`*`/ggc/var/log/system/GGStreamManager.log`.

Si necesita más información sobre la solución de problemas, puede establecer el nivel de registro para Registros de usuario de Lambda en Registros de depuración y, a continuación, implementar el grupo de nuevo.

Véase también

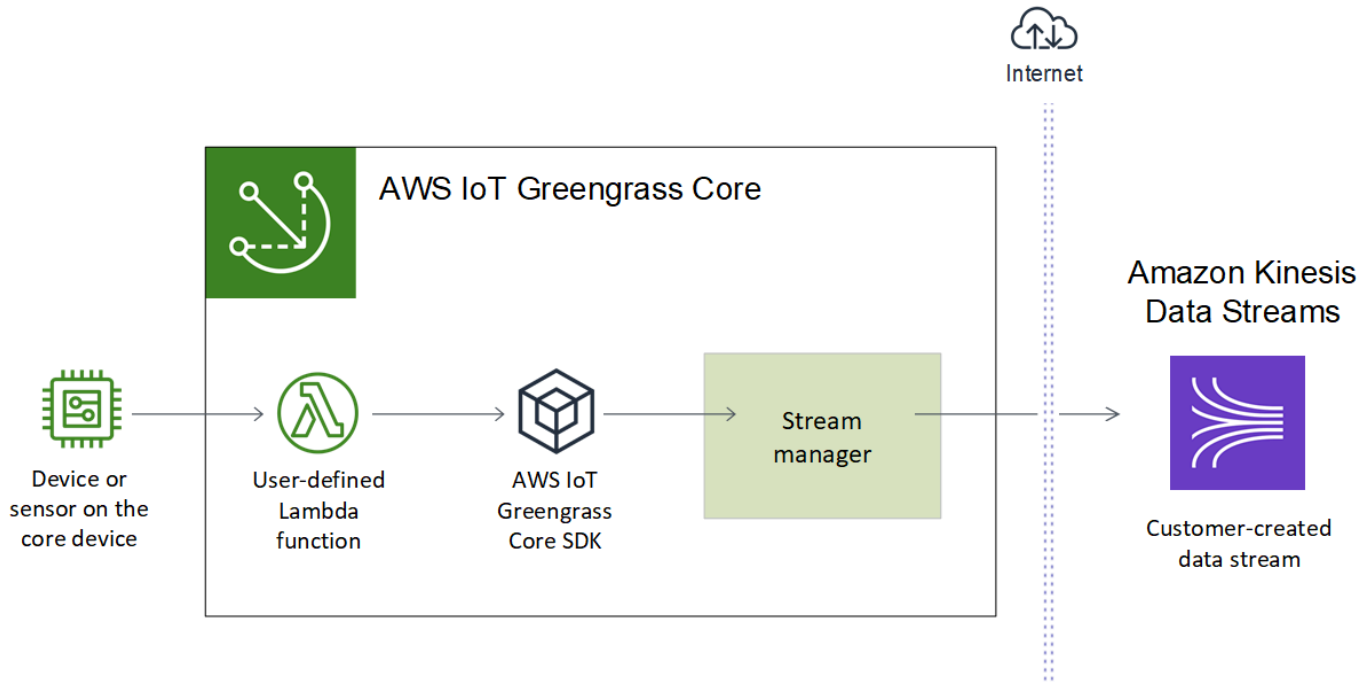
- Administrar secuencias de datos
- the section called “Configurar el administrador de secuencias”
- the section called “Utilizar StreamManagerClient para trabajar con secuencias”
- the section called “Exportación de configuraciones para Nube de AWS destinos compatibles”
- the section called “Exportar secuencias de datos (CLI)”

Exportar secuencias de datos a la Nube de AWS (CLI)

En este tutorial se muestra cómo utilizar el AWS CLI para configurar e implementar un grupo de AWS IoT Greengrass con el gestor de secuencias habilitado. El grupo contiene una función de Lambda definida por el usuario que escribe en una secuencia en el administrador de secuencias, que luego se exporta automáticamente a la Nube de AWS.

El administrador de secuencias hace que la asimilación, el procesamiento y la exportación de flujos de datos de gran volumen sea más eficiente y fiable. En este tutorial, va a crear una función de

Lambda de `TransferStream` que consume datos de IoT. La función de Lambda utiliza el SDK de AWS IoT Greengrass Core para crear una secuencia en el administrador de secuencias y luego leer y escribir en ella. El administrador de secuencias exporta la secuencia al Flujo de datos Kinesis. En el siguiente diagrama se muestra este flujo de trabajo.



El objetivo de este tutorial es mostrar cómo utilizan las funciones de Lambda definidas por el usuario el objeto de `StreamManagerClient` en el SDK de AWS IoT Greengrass Core para interactuar con el administrador de flujos. Para simplificar, la función de Lambda que va a crear en este tutorial genera datos de dispositivos simulados.

Cuando utilice la API de AWS IoT Greengrass, que incluye los comandos de Greengrass en la AWS CLI, para crear un grupo, el gestor de flujos estará desactivado por defecto. Para habilitar el administrador de secuencias en su núcleo, cree una versión de definición de función que incluya la función de Lambda del sistema `GGStreamManager` y una versión de grupo que haga referencia a la nueva versión de definición de función. A continuación, implemente el grupo.

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Un grupo de Greengrass y un núcleo de Greengrass (versión 1.10 o posterior). Para obtener información acerca de cómo crear un núcleo y un grupo de Greengrass, consulte Empezando con

AWS IoT Greengrass. El tutorial de introducción también incluye pasos para instalar el software AWS IoT Greengrass Core.

> **Note**
>
> El administrador de secuencias no es compatible con las distribuciones OpenWrt.

- Java 8 Runtime (JDK 8) instalado en el dispositivo principal.
  - Para distribuciones basadas en Debian (incluido Raspbian) o distribuciones basadas en Ubuntu, ejecute el siguiente comando:

    ```
    sudo apt install openjdk-8-jdk
    ```

  - Para distribuciones basadas en Red Hat (incluido Amazon Linux), ejecute el siguiente comando:

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Para obtener más información, consulte How to download and install prebuilt OpenJDK packages (Cómo descargar e instalar paquetes OpenJDK preconfigurados) en la documentación de OpenJDK.
- SDK de AWS IoT Greengrass Core para Python versión 1.5.0 o versiones posteriores Para usar `StreamManagerClient` en el SDK de AWS IoT Greengrass Core para Python, debe:
  - Instalar Python 3.7 o versiones posteriores en el dispositivo principal.
  - Incluya el SDK y sus dependencias en su paquete de implementación de la función de Lambda. Se incluyen las instrucciones en este tutorial.

> **Tip**
>
> Puede usar `StreamManagerClient` con Java o NodeJS. Para ver código de ejemplo, consulte el SDK de AWS IoT Greengrass Core para Java y el SDK de AWS IoT Greengrass Core para Node.js en GitHub.

- Una secuencia de destino denominada **`MyKinesisStream`** creada en Amazon Kinesis Data Streams en la misma Región de AWS que su grupo de Greengrass. Para obtener más información, consulte Crear un flujo en la Guía para desarrolladores de Amazon Kinesis.

> **Note**
>
> En este tutorial, el administrador de secuencias exporta datos Kinesis Data Streams, lo que deriva en cargos a su Cuenta de AWS. Para obtener información acerca de los precios, consulte Precios de Kinesis Data Streams.
> Para no incurrir en gastos, puede ejecutar este tutorial sin crear una secuencia de datos Kinesis. En este caso, compruebe los registros para ver si el administrador de flujos intentó exportar la secuencia al flujo de datos Kinesis.

- Una política de IAM agregada al the section called “Rol de grupo de Greengrass” que permita la acción de `kinesis:PutRecords` en el flujo de datos de destino, tal y como se muestra en el siguiente ejemplo:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

- La AWS CLI instalada y configurada en su ordenador. Para obtener más información, consulte Instalación de la AWS Command Line Interface y Configuración de la AWS CLI en la Guía del usuario de la AWS Command Line Interface.

  Los comandos de ejemplo de este tutorial están escritos para Linux y otros sistemas basados en Unix. Si utiliza Windows, consulte Especificación de valores de parámetros para la interfaz de línea de comandos AWS para obtener más información sobre las diferencias de sintaxis.

Si el comando contiene una cadena JSON, el tutorial ofrece un ejemplo que tiene el JSON en una sola línea. En algunos sistemas, puede que sea más eficiente editar y ejecutar comandos con este formato.

El tutorial contiene los siguientes pasos generales:

1. Creación de un paquete de implementación de la función de Lambda
2. Creación de una función de Lambda
3. Crear una versión y una definición de la función
4. Crear una versión y una definición del registrador
5. Obtener el ARN de la versión de la definición del núcleo
6. Cree una versión del grupo
7. Crear una implementación
8. Pruebe la aplicación.

Completar el tutorial debería tomarle aproximadamente 30 minutos.

Paso 1: Crear un paquete de implementación de la función de Lambda

En este paso, va a crear un paquete de implementación de funciones de Lambda que contiene código de función y dependencias de Python. Cargará este paquete más adelante cuando cree la función de Lambda en AWS Lambda. La función de Lambda utiliza el SDK de AWS IoT Greengrass Core para crear secuencias locales e interactuar con ellas.

> **Note**
>
> Sus funciones de Lambda definidas por el usuario deben utilizar el SDK de AWS IoT Greengrass Core para interactuar con el administrador de flujo. Para obtener más información sobre los requisitos del administrador de secuencias de Greengrass, consulte Requisitos del administrador de secuencias de Greengrass.

1. Descargue el SDK de AWS IoT Greengrass Core para Python versión 1.5.0 o posterior.

2. Descomprima el paquete descargado para obtener el SDK. El SDK es la carpeta `greengrasssdk`.
3. Instale dependencias de paquetes para incluirlas con el SDK en su paquete de implementación de funciones de Lambda.

   1. Vaya al directorio de SDK que contiene el archivo de `requirements.txt`. Este archivo registra las dependencias.
   2. Instale las dependencias del SDK. Por ejemplo, ejecute el siguiente comando de `pip` para instalarlas en el directorio actual:

      ```
      pip install --target . -r requirements.txt
      ```

4. Guarde la siguiente función de código de Python en un archivo local llamado "`transfer_stream.py`".

   > **Tip**
   >
   > Para ver un ejemplo de código que utiliza Java y NodeJS, consulte el SDK de AWS IoT Greengrass Core for Java y SDK de AWS IoT Greengrass Core para Node.js en GitHub.

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
 exports
```

```
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
 kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
 strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
```

```
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
 ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
 to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
 sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
 1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. Comprima en un archivo ZIP los siguientes elementos en un archivo denominado "`transfer_stream_python.zip`". Este es el paquete de implementación de la función de Lambda.

   - transfer_stream.py. Lógica de la aplicación.
   - greengrasssdk. Biblioteca necesaria para las funciones de Lambda de Python Greengrass que publican mensajes MQTT.

Las operaciones del administrador de secuencias están disponibles en la versión 1.5.0 o en las versiones posteriores del SDK de AWS IoT Greengrass Core para Python.

- Las dependencias que instaló para el SDK de AWS IoT Greengrass Core para Python (por ejemplo, los directorios de `cbor2`).

Al crear el archivo de `zip`, incluya solo estos elementos, no la carpeta que los contiene.

Paso 2: crear una función Lambda

1. Cree un rol de IAM para poder pasar el ARN del rol cuando cree la función.

   JSON Expanded

   ```
   aws iam create-role --role-name Lambda_empty --assume-role-policy '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "lambda.amazonaws.com"
               },
              "Action": "sts:AssumeRole"
           }
       ]
   }'
   ```

   JSON Single-line

   ```
   aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
    "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
    "lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
   ```

> **Note**
>
> AWS IoT Greengrass no utiliza este rol, ya que los permisos de las funciones de Lambda de Greengrass se especifican en la función de rol de grupo de Greengrass. En este tutorial, va a crear un rol vacío.

2. Copie la `Arn` del resultado.
3. Utilice la API de AWS Lambda para crear la función de `TransferStream`. El siguiente comando da por hecho que el archivo ZIP está en el directorio actual.

   - Reemplace *`role-arn`* por el `Arn` que ha copiado.

```
aws lambda create-function \
--function-name TransferStream \
--zip-file fileb://transfer_stream_python.zip \
--role role-arn \
--handler transfer_stream.function_handler \
--runtime python3.7
```

4. Publique una versión de la función.

```
aws lambda publish-version --function-name TransferStream --description 'First
 version'
```

5. Cree un alias a la versión publicada.

   Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --
function-version 1
```

> **Note**
>
> AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST.

6. Copie la `AliasArn` del resultado. Este valor se usa al configurar la función para AWS IoT Greengrass.

Ahora está listo para configurar la función para AWS IoT Greengrass.

Paso 3: Crear una versión y una definición de la función

Este paso crea una versión de definición de función que hace referencia a la función de Lambda `GGStreamManager` del sistema y a la función Lambda `TransferStream` definida por el usuario. Para habilitar el administrador de secuencias al usar la API AWS IoT Greengrass, la versión de definición de funciones debe incluir la función `GGStreamManager`.

1. Cree una definición de función con una versión inicial que contenga las funciones de Lambda del sistema y definidas por el usuario.

   Con la siguiente versión de definición se habilita al administrador de secuencias con la configuración de parámetros predeterminada. Para configurar parámetros personalizados, debe definir variables de entorno para los parámetros correspondientes del administrador de secuencias. Para ver un ejemplo, consulte the section called “Habilitar, deshabilitar o configurar el administrador de secuencias”. AWS IoT Greengrass utiliza características predeterminadas para los parámetros que se omiten. `MemorySize` debería ser al menos `128000`. `Pinned` debe establecerse en `true`.

   > **Note**
   >
   > Una función de Lambda de larga duración (o anclada) se inicia automáticamente después de arrancar AWS IoT Greengrass y sigue ejecutándose en su propio contenedor. Esto contrasta con una función de Lambda bajo demanda, que se inicia cuando se la invoca y se detiene cuando no quedan tareas que ejecutar. Para obtener más información, consulte the section called “Configuración del ciclo de vida”.

- Reemplace *id-función-arbitraria* con un nombre para la función, como, por ejemplo, **stream-manager**.
- Reemplace *alias-arn* por el `AliasArn` que ha copiado al crear el alias para la función de Lambda de `TransferStream`.

JSON expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "arbitrary-function-id",
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize": 128000,
                "Pinned": true,
                "Timeout": 3
            }
        },
        {
            "Id": "TransferStreamFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
                "Executable": "transfer_stream.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON single

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "arbitrary-function-
id","FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
```

```
 "FunctionConfiguration": {"Environment": {"Variables":
{"STREAM_MANAGER_STORE_ROOT_DIR": "/data","STREAM_MANAGER_SERVER_PORT":
 "1234","STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH": "20000"}},"MemorySize":
 128000,"Pinned": true,"Timeout": 3}},{"Id": "TransferStreamFunction",
 "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
 "transfer_stream.function_handler", "MemorySize": 16000,"Pinned":
 true,"Timeout": 5}}]}'
```

> **Note**
>
> `Timeout` es necesario para la versión de definición de función, pero `GGStreamManager` no lo usa. Para obtener más información sobre `Timeout` y otras configuraciones a nivel de grupo, consulte the section called “Control de la ejecución de la función de Lambda de Greengrass”.

2. Copie la `LatestVersionArn` del resultado. Este valor se usa para añadir la versión de la definición de función a la versión de grupo que implementó en el núcleo.

Paso 4: Crear una versión y una definición del registrador

Defina la configuración de registro del grupo. Para este tutorial, debe configurar los componentes del sistema AWS IoT Greengrass, las funciones de Lambda definidas por el usuario y los conectores para que escriban los registros en el sistema de archivos del dispositivo principal. Puede usar registros para solucionar cualquier problema que pueda surgir. Para obtener más información, consulte the section called “Monitorización con registros de AWS IoT Greengrass”.

1. Cree una definición del registro que incluya una versión inicial.

   JSON Expanded

```
aws greengrass create-logger-definition --name "LoggingConfigs" --initial-
version '{
    "Loggers": [
        {
            "Id": "1",
            "Component": "GreengrassSystem",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
```

```
        },
        {
            "Id": "2",
            "Component": "Lambda",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-logger-definition \
    --name "LoggingConfigs" \
    --initial-version '{"Loggers":
[{"Id":"1","Component":"GreengrassSystem","Level":"INFO","Space":10240,"Type":"FileSyste
{"Id":"2","Component":"Lambda","Level":"INFO","Space":10240,"Type":"FileSystem"}]}'
```

2. Copie el `LatestVersionArn` de la definición del registro del resultado. Este valor se usa para añadir la versión de definición del registro a la versión del grupo que implementa en el núcleo.

Paso 5: Obtener el ARN de la versión de la definición del núcleo

Obtenga el ARN de la versión de definición del núcleo para agregar a su nueva versión de grupo. Para implementar una versión de grupo, debe hacer referencia a una versión de definición de núcleo que contenga exactamente un núcleo.

1. Obtenga los ID de la versión de grupo y grupo de Greengrass de destino. En este procedimiento, suponemos que estos son el último grupo y la última versión de grupo. La siguiente consulta devuelve el grupo creado más recientemente.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

También puede hacer la consulta por nombre. No es necesario que los nombres de grupo sean únicos, por lo que podrían devolverse varios grupos.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

> **Note**
>
> También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

2. Copie el `Id` del grupo de destino de la salida. Puede utilizar esto para obtener la versión de la definición de núcleo y al implementar el grupo.
3. Copie el `LatestVersion` del resultado, que es el ID de la última versión añadida al grupo. Puede utilizar esto para obtener la versión de la definición de núcleo.
4. Obtenga el ARN de la versión de la definición principal:

   a. Obtenga la versión de grupo.

   - Reemplace *`id-grupo`* con el `Id` que ha copiado para el grupo.
   - Reemplace *`group-version-id`* por el parámetro `LatestVersion` que ha copiado para el grupo.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id group-version-id
   ```

   b. Copie la `CoreDefinitionVersionArn` del resultado. Este valor se usa para añadir la versión de la definición del núcleo a la versión de grupo que implementó en el núcleo.

Paso 6: Crear una versión del grupo

Ahora, puede crear una versión de grupo que contenga las entidades que desea implementar. Para ello, cree una versión de grupo que haga referencia a la versión de destino de cada tipo de componente. Para este tutorial, incluirá una versión de definición de núcleo, una versión de definición de función y una versión de definición de registrador.

1. Cree una versión de grupo.

   - Reemplace *`id-grupo`* con el `Id` que ha copiado para el grupo.

- Reemplace *core-definition-version-arn* por el `CoreDefinitionVersionArn` que ha copiado para la nueva versión de la definición de núcleo.
- Reemplace *function-definition-version-arn* con el `LatestVersionArn` que ha copiado para su nueva versión de definición de la función.
- Reemplace *logger-definition-version-arn* con el `LatestVersionArn` que copió para la nueva versión de definición del registrador.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn
```

2. Copie la `Version` del resultado. Este es el ID de la nueva versión del grupo.

Paso 7: Crear una implementación

Implemente el grupo en el dispositivo del núcleo.

1. Asegúrese de que el núcleo de AWS IoT Greengrass se está ejecutando. Ejecute los siguientes comandos en el terminal de Raspberry Pi según sea necesario.

   a. Para comprobar si el demonio está en ejecución:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/`*ggc-version*`/bin/daemon`, el demonio está en ejecución.

   > **Note**
   >
   > La versión que figura en la ruta depende de la versión del software AWS IoT Greengrass Core que esté instalada en el dispositivo del núcleo.

   b. Iniciar el daemon:

   ```
   cd /greengrass/ggc/core/
   ```

```
sudo ./greengrassd start
```

2. Cree una implementación de .

   - Reemplace *id-grupo* con el `Id` que ha copiado para el grupo.
   - Reemplace *id-versión-grupo* con el `Version` que ha copiado para el nuevo grupo.

```
aws greengrass create-deployment \
--deployment-type NewDeployment \
--group-id group-id \
--group-version-id group-version-id
```

3. Copie la `DeploymentId` del resultado.
4. Obtenga el estado de las implementaciones.

   - Reemplace *id-grupo* con el `Id` que ha copiado para el grupo.
   - Reemplace *deployment-id* por el parámetro `DeploymentId` que ha copiado para la implementación.

```
aws greengrass get-deployment-status \
--group-id group-id \
--deployment-id deployment-id
```

   Si el estado es `Success`, la implementación fue correcta. Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

Paso 8: Probar la aplicación

La función de Lambda `TransferStream` genera datos simulados del dispositivo. Escribe datos en una secuencia que el administrador de secuencias exporta a la secuencia de datos de Kinesis de destino.

1. En la consola de Amazon Kinesis , bajo las Secuencias de datos de Kinesis, seleccione MyKinesisStream.

> **Note**
>
> Si ejecutó el tutorial sin una secuencia de datos de Kinesis de destino, compruebe el archivo de registro del administrador de secuencias (`GGStreamManager`). Si contiene `export stream MyKinesisStream doesn't exist` en un mensaje de error, la prueba se ha realizado correctamente. Este error significa que el servicio intentó exportar a la secuencia, pero que la secuencia no existe.

2. En la página MyKinesisStream, seleccione Monitoring (Supervisión). Si la prueba se realiza correctamente, debería ver los datos en los gráficos Put Records. En función de la conexión, es posible que tarde un minuto en mostrar los datos.

> **Important**
>
> Cuando haya terminado la prueba, elimine la secuencia de datos de Kinesis para evitar incurrir en más gastos.
> O ejecute el siguiente comando para detener el daemon de Greengrass. Así evitará que el núcleo envíe mensajes hasta que esté listo para continuar las pruebas.
>
> ```
> cd /greengrass/ggc/core/
> sudo ./greengrassd stop
> ```

3. Elimine la función de Lambda TransferStream del núcleo.

   a. Siga the section called “Cree una versión del grupo” para crear una nueva versión de grupo. pero elimine la opción `--function-definition-version-arn` en el comando `create-group-version`. O bien, cree una versión de definición de función que no incluya la función de Lambda TransferStream.

   > **Note**
   >
   > Al omitir la función de Lambda de `GGStreamManager` del sistema de la versión de grupo implementada, se deshabilita la administración de secuencias en el núcleo.

   b. Siga the section called “Crear una implementación” para implementar la nueva versión de grupo.

Para ver la información de registro o solucionar problemas con las secuencias, compruebe los registros para las funciones `TransferStream` y `GGStreamManager`. Debe tener permisos de `root` para leer registros AWS IoT Greengrass en el sistema de archivos.

- `TransferStream` escribe entradas de registro en *greengrass-root*`/ggc/var/log/user/`*region*`/`*account-id*`/TransferStream.log`.
- `GGStreamManager` escribe entradas de registro en *greengrass-root*`/ggc/var/log/system/GGStreamManager.log`.

Si necesita más información sobre la solución de problemas, puede establecer el nivel de registro de `Lambda` a `DEBUG` y, a continuación, crear e implementar una nueva versión de grupo.

Véase también

- Administrar secuencias de datos
- the section called “Utilizar StreamManagerClient para trabajar con secuencias”
- the section called “Exportación de configuraciones para Nube de AWS destinos compatibles”
- the section called “Configurar el administrador de secuencias”
- the section called “Exportar flujos de datos (consola)”
- Comandos de AWS Identity and Access Management (IAM) en la Referencia de comandos de la AWS CLI.
- Comandos de AWS Lambda en la Referencia de comandos de AWS CLI
- Comandos de AWS IoT Greengrass en la Referencia de comandos de AWS CLI

Implementación de secretos en el núcleo de AWS IoT Greengrass

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y versiones posteriores.

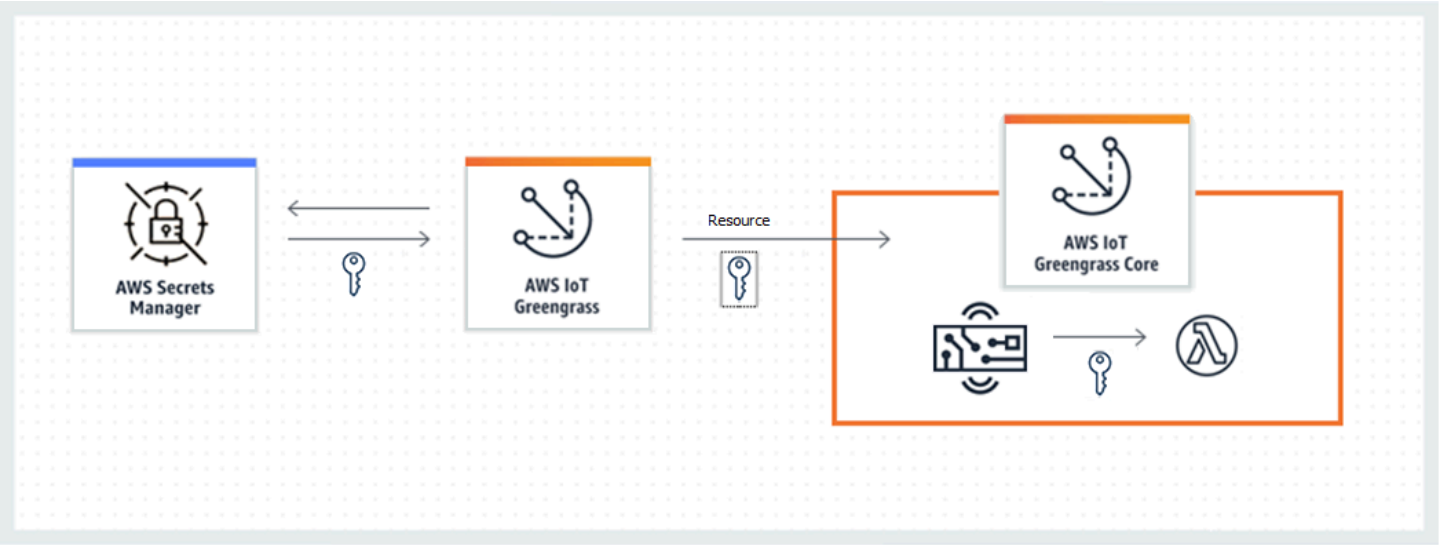
AWS IoT Greengrass le permite autenticarse con servicios y aplicaciones desde dispositivos Greengrass sin contraseñas codificadas de forma rígida, tokens u otros secretos.

AWS Secrets Manager es un servicio que puede usar para almacenar y administrar los secretos de forma segura en la nube. AWS IoT Greengrass amplía Secrets Manager a dispositivos del núcleo Greengrass, de modo que los conectores y las funciones de Lambda puedan utilizar secretos locales para interactuar con los servicios y aplicaciones. Por ejemplo, el conector de notificaciones de Twilio utiliza un token de autenticación almacenado localmente.

Para integrar un secreto en un grupo de Greengrass, cree un recurso de grupo que haga referencia al secreto de Secrets Manager. Este recurso de secretos hace referencia al secreto en la nube por ARN. Para aprender a crear, administrar y utilizar recursos de secretos, consulte the section called “Uso de recursos de secretos”.

AWS IoT Greengrass cifra los secretos en tránsito y en reposo. Durante la implementación del grupo, AWS IoT Greengrass recupera el secreto desde Secrets Manager y crea una copia cifrada, local en el núcleo de Greengrass. Después de rotar los secretos en la nube en Secrets Manager, vuelva a implementar el grupo para propagar los valores actualizados en el núcleo.

En el siguiente diagrama se muestra el proceso general de implementación de un secreto en el núcleo. Los secretos se cifran en reposo y en tránsito.



Al usar AWS IoT Greengrass para almacenar los secretos localmente se obtienen estas ventajas:

- Se desacoplan del código (sin codificar de forma rígida). Es compatible con las credenciales administradas de manera centralizada y ayuda a proteger la información confidencial para que no se filtre.
- Disponible para situaciones sin conexión Los conectores y las funciones pueden acceder de manera segura al software y a los servicios locales cuando están desconectados de Internet.
- Acceso controlado a los secretos. Solo las funciones y los conectores autorizados en el grupo pueden acceder a los secretos. AWS IoT Greengrass utiliza cifrado de clave privada para proteger los secretos. Los secretos se cifran en reposo y en tránsito. Para obtener más información, consulte the section called “Cifrado de secretos”.
- Rotación controlada. Después de rotar los secretos en Secrets Manager, vuelva a implementar el grupo de Greengrass para actualizar la copias locales de los secretos. Para obtener más información, consulte the section called “Creación y administración de secretos”.

> **Important**
>
> AWS IoT Greengrass no actualiza automáticamente los valores de los secretos locales después de rotar las versiones en la nube. Para actualizar los valores locales, debe volver a implementar el grupo.

Cifrado de secretos

AWS IoT Greengrass cifra los secretos en reposo y en tránsito.

> **Important**
>
> Asegúrese de que las funciones de Lambda definidas por el usuario gestionen los secretos de forma segura y no registren ningún dato confidencial almacenado en el secreto. Para obtener más información, consulte Mitigar los riesgos de registrar y depurar la función de Lambda en la Guía del usuario de AWS Secrets Manager. Si bien esta documentación se refiere específicamente a las funciones de rotación, la recomendación también se aplica a las funciones de Lambda de Greengrass.

Cifrado en tránsito

AWS IoT Greengrass utiliza Transport Layer Security (TLS) para cifrar toda la comunicación a través de Internet y la red local. De esta forma se protegen los secretos en tránsito, que se produce cuando se recuperan los secretos de Secrets Manager y se implementan en el núcleo. Para ver conjuntos de cifrado TLS compatibles, consulte the section called “Compatibilidad con conjuntos de cifrado TLS”.

Cifrado en reposo

AWS IoT Greengrass utiliza la clave privada especificada en `config.json` para cifrado de los secretos que se almacenan en el núcleo. Por este motivo, el almacenamiento seguro de la clave privada es fundamental para proteger secretos locales. En el modelo de responsabilidad compartida de AWS, es responsabilidad del cliente garantizar el almacenamiento seguro de la clave privada en el dispositivo principal.

AWS IoT Greengrass admite dos modos de almacenamiento de claves privadas:

- Uso de módulos de seguridad de hardware. Para obtener más información, consulte the section called “Integración de la seguridad de hardware”.

  > **Note**
  >
  > Actualmente, solo AWS IoT Greengrass admite el mecanismo de relleno PKCS #1 versión 1.5 para el cifrado y descifrado de secretos locales cuando se utilizan claves privadas basadas en hardware. Si sigue las instrucciones del proveedor para generar

manualmente claves privadas basadas en hardware, asegúrese de elegir PKCS #1 versión 1.5. AWS IoT Greengrassno es compatible con el relleno de cifrado asimétrico óptimo (OAEP).

- Uso de permisos del sistema de archivos (predeterminado).

La clave privada se utiliza para proteger la clave de datos, que se utiliza para cifrar secretos locales. La clave de datos se rota con cada implementación de grupo.

El núcleo de AWS IoT Greengrass es la única entidad que tiene acceso a la clave privada. Los conectores de Greengrass o las funciones de Lambda afiliados a un recurso de secreto obtienen el valor del secreto del núcleo.

Requisitos

Estos son los requisitos para admitir los secretos locales:

- Debe utilizar Núcleo de AWS IoT Greengrass versión 1.7 o posterior.
- Para obtener los valores de los secretos locales, las funciones de Lambda definidas por el usuario deben usar el SDK de AWS IoT Greengrass Core versión 1.3.0 o una versión posterior.
- La clave privada utilizada para el cifrado de secretos locales deben especificarse en el archivo de configuración de Greengrass. De forma predeterminada, AWS IoT Greengrass utiliza la clave privada del núcleo almacenada en el sistema de archivos. Para proporcionar su propia clave privada, consulte the section called “Especificación de la clave privada para el cifrado de secretos”. Solo se admite el tipo de claves RSA.

> **Note**
>
> Actualmente, solo AWS IoT Greengrass admite el mecanismo de relleno PKCS #1 versión 1.5 para el cifrado y descifrado de secretos locales cuando se utilizan claves privadas basadas en hardware. Si sigue las instrucciones del proveedor para generar manualmente claves privadas basadas en hardware, asegúrese de elegir PKCS #1 versión 1.5. AWS IoT Greengrassno es compatible con el relleno de cifrado asimétrico óptimo (OAEP).

- AWS IoT Greengrass debe tener permiso para obtener los valores de secretos. De este modo, AWS IoT Greengrass puede recuperar los valores durante la implementación del grupo. Si está utilizando el rol de servicio de Greengrass predeterminado, AWS IoT Greengrass ya tendrá acceso

a los secretos con nombres que comiencen por greengrass-. Para personalizar el acceso, consulte the section called “Permitir que AWS IoT Greengrass obtenga valores de secretos”.

> **Note**
>
> Le recomendamos que utilice esta convención de nomenclatura para identificar los secretos a los que AWS IoT Greengrass puede acceder, aunque personalice los permisos. La consola utiliza diferentes permisos para leer los secretos, por lo que es posible que puede seleccionar secretos en la consola que AWS IoT Greengrass no tiene permiso para recuperar. El uso de una convención de nomenclatura puede ayudarle a evitar un permiso conflicto, lo que provoca un error de implementación.

Especificación de la clave privada para el cifrado de secretos

En este procedimiento, debe proporcionar la ruta a una clave privada que se utiliza para el cifrado secreto local. Debe ser una clave RSA con una longitud mínima de 2048 bits. Para obtener más información acerca de las claves privadas utilizadas en el AWS IoT Greengrass del núcleo, consulte the section called “Entidades de seguridad”.

AWS IoT Greengrass admite dos modos de almacenamiento de claves privadas: basado en hardware o basado en el sistema de archivos (valor predeterminado). Para obtener más información, consulte the section called “Cifrado de secretos”.

Siga este procedimiento solo si desea cambiar la configuración predeterminada, que utiliza la clave privada del núcleo en el sistema de archivos. Estos pasos se han redactado partiendo del supuesto de que creó el grupo y el núcleo tal y como se describe en el módulo 2 del tutorial de introducción.

1. Abra el archivo `config.json` situado en el directorio `/greengrass-root/config`.

   > **Note**
   >
   > *`greengrass-root`* representa la ruta donde está instalado el software de AWS IoT Greengrass Core en su dispositivo. Normalmente, este es el directorio `/greengrass`.

2. En el objeto `crypto.principals.SecretsManager` de la propiedad `privateKeyPath`, escriba la ruta de la clave privada:

- Si su clave privada se almacena en el sistema de archivos, especifique la ruta absoluta a la clave. Por ejemplo:

```
"SecretsManager" : {
  "privateKeyPath" : "file:///somepath/hash.private.key"
}
```

- Si la clave privada se almacena en un módulo de seguridad de hardware (HSM), especifique la ruta con el esquema de URI RFC 7512 PKCS#11. Por ejemplo:

```
"SecretsManager" : {
  "privateKeyPath" : "pkcs11:object=private-key-label;type=private"
}
```

Para obtener más información, consulte the section called “Configuración de la seguridad de hardware”.

> **Note**
>
> Actualmente, solo AWS IoT Greengrass admite el mecanismo de relleno PKCS #1 versión 1.5 para el cifrado y descifrado de secretos locales cuando se utilizan claves privadas basadas en hardware. Si sigue las instrucciones del proveedor para generar manualmente claves privadas basadas en hardware, asegúrese de elegir PKCS #1 versión 1.5. AWS IoT Greengrassno es compatible con el relleno de cifrado asimétrico óptimo (OAEP).

Permitir que AWS IoT Greengrass obtenga valores de secretos

En este procedimiento añadirá una política en línea al rol de servicio de Greengrass que permite que AWS IoT Greengrass obtenga los valores de los secretos.

Siga este procedimiento solo si desea conceder a AWS IoT Greengrass permisos personalizados a los secretos o si el rol de servicio de Greengrass no incluye la `AWSGreengrassResourceAccessRolePolicy` política administrada. `AWSGreengrassResourceAccessRolePolicy` concede acceso a los secretos con nombres que comiencen por greengrass-.

1. Ejecute el siguiente comando de la CLI para obtener el ARN del rol de servicio de Greengrass:

```
aws greengrass get-service-role-for-account --region region
```

El ARN que se devuelve contendrá el nombre del rol.

```
{
  "AssociatedAt": "time-stamp",
  "RoleArn": "arn:aws:iam::account-id:role/service-role/role-name"
}
```

En el siguiente paso usará el ARN o el nombre.

2. Añada una política en línea que permita la acción `secretsmanager:GetSecretValue`. Para obtener instrucciones, consulte Añadir y eliminar políticas de IAM en la Guía del usuario de IAM.

Puede conceder acceso detallado enumerando explícitamente los secretos o usando un esquema de nomenclatura *, o bien puede conceder acceso condicional a secretos con control de versiones o etiquetados. Por ejemplo, la siguiente política permite a AWS IoT Greengrass leer solo los secretos especificados.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:region:account-id:secret:greengrass-
SecretA-abc",
                "arn:aws:secretsmanager:region:account-id:secret:greengrass-
SecretB-xyz"
            ]
        }
    ]
}
```

> **Note**
>
> Si utiliza una clave de AWS KMS administrada por el cliente para cifrar secretos, el rol de servicio de Greengrass también debe permitir la acción `kms:Decrypt`.

Para obtener más información sobre las políticas de IAM para Secrets Manager, consulte Autenticación y control de acceso para AWS Secrets Manager y Acciones, recursos y claves de contexto que puede utilizar en una política de IAM o política secreta para AWS Secrets Manager en la Guía del usuario de AWS Secrets Manager.

Véase también

- ¿Qué es AWS Secrets Manager? en la Guía del usuario de AWS Secrets Manager
- PKCS #1: Cifrado RSA versión 1.5

Uso de recursos de secretos

AWS IoT Greengrass utiliza recursos de secretos para integrar secretos desde AWS Secrets Manager en un grupo de Greengrass. Un recurso de secreto es una referencia a un secreto de Secrets Manager. Para obtener más información, consulte Implementación de secretos en el núcleo.

En el dispositivo AWS IoT Greengrass principal, los conectores y las funciones de Lambda pueden usar el recurso de secretos para autenticarse con servicios y aplicaciones, sin necesidad de codificación rígida de contraseñas, tokens u otras credenciales.

Creación y administración de secretos

En un grupo Greengrass, un recurso secreta hace referencia al ARN de un secreto de Secrets Manager. Cuando el recurso de secretos se implementa en el núcleo, el valor del secreto se cifra y se pone a disposición de los conectores afiliados y las funciones Lambda. Para obtener más información, consulte the section called “Cifrado de secretos”.

Puede utilizar Secrets Manager para crear y administrar las versiones en la nube de los secretos. Puede usar AWS IoT Greengrass para crear, administrar e implementar recursos de secretos.

> **Important**
>
> Le recomendamos que siga la práctica recomendada de rotación de secretos en Secrets Manager. A continuación, implemente el grupo de Greengrass para actualizar las copias locales de los secretos. Para obtener más información, consulte Rotación de sus secretos de AWS Secrets Manager en la guía del usuario de AWS Secrets Manager.

Para poner un secreto a disposición del núcleo de Greengrass, siga estos pasos:

1. Cree un secreto en Secrets Manager. Esta es la versión en la nube del secreto, que se almacena y administra de forma centralizada en Secrets Manager. Entre las tareas de administración se incluyen la rotación de valores de secretos y la aplicación de políticas de recursos.
2. Cree un recurso de secretos en AWS IoT Greengrass. Se trata de un tipo de recurso de grupo que hace referencia al secreto en la nube por ARN. Solo puede hacer referencia a un secreto por grupo.
3. Configure la función de Lambda con la consola o la función de Lambda. Debe afiliar el recurso con un conector o una función especificando los parámetros o las propiedades correspondientes. Esto les permite obtener el valor del recurso de secretos implementado localmente. Para obtener más información, consulte the section called “Uso de secretos locales”.
4. Implemente el grupo de Greengrass. Durante la implementación, AWS IoT Greengrass recupera el valor del secreto en la nube y crea secreta (o actualiza) el secreto locales en el núcleo.

Secrets Manager registra un evento en AWS CloudTrail cada vez que AWS IoT Greengrass recupera un valor de secretos. AWS IoT Greengrass no registra los eventos relacionados con la implementación o el uso de secretos locales. Para obtener información sobre las llamadas registradas por Secrets Manager, consulte Monitorización del uso de sus secretos AWS Secrets Manager en la Guía de usuario de AWS Secrets Manager.

Incluir etiquetas de ensayo en recursos de secretos

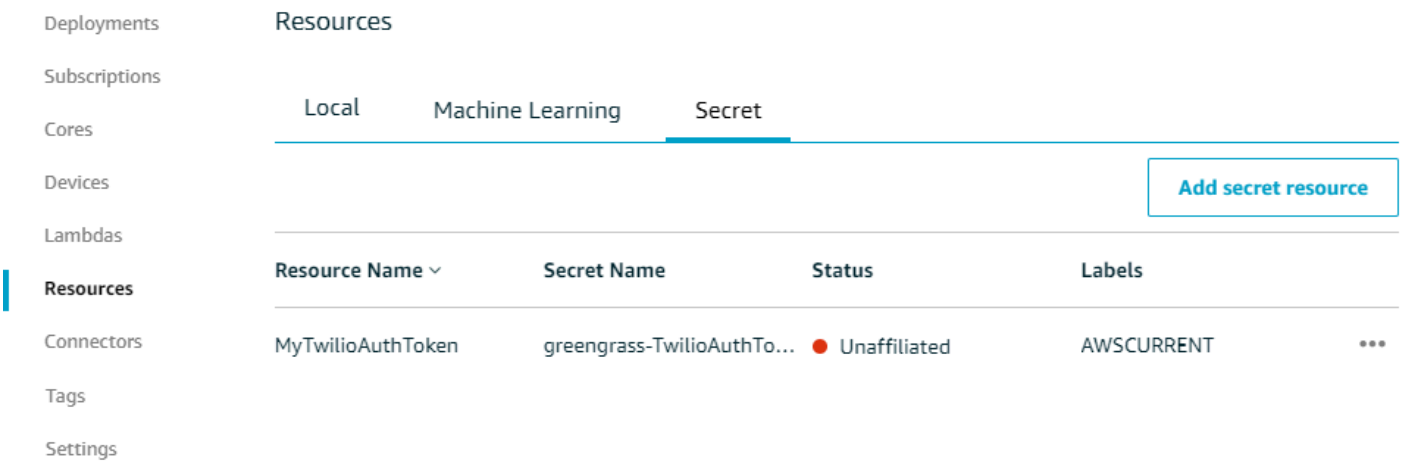
Secrets Manager utiliza etiquetas de ensayo para especificar las versiones de un valor de secretos. Las etiquetas de preparación pueden estar definidas por el sistema o definidas por el usuario. Secrets Manager asigna la etiqueta `AWSCURRENT` a la versión más reciente del valor del secreto. Las etiquetas de ensayo se utilizan habitualmente para administrar la rotación de secretos. Para obtener más información acerca del control de versiones de los secretos, consulte los Términos y conceptos clave de AWS Secrets Manager en la Guía del usuarioAWS Secrets Manager.

Los recursos de secretos siempre incluyen la etiqueta de ensayo `AWSCURRENT` y también pueden incluir otras etiquetas de ensayo si las requiere una función de Lambda o un conector. Durante la implementación del grupo, AWS IoT Greengrass recupera los valores de las etiquetas de ensayo a los que se hace referencia en el grupo y, a continuación, crea o actualiza los valores correspondientes en el núcleo.

Creación y administración de recursos de secretos (consola)

Creación de recursos de secretos (consola)

En el AWS IoT Greengrass, puede crear y administrar recursos de secretos en la pestaña Secretos de la página Recursos del grupo. Para ver tutoriales que crean un recurso de secretos y lo añaden a un grupo, consulte the section called “Cómo crear un recurso de secreto (consola)” y the section called “Introducción a los conectores (consola)”.



> **Note**
>
> Como alternativa, la consola le permite crear un secreto y un recurso secreto al configurar un conector o una función de Lambda. Puede hacerlo desde la página Configurar parámetros del conector o desde la página Recursos de la función de Lambda.

Administración de recursos de secretos (consola)

Las tareas de administración de los recursos de secretos de su grupo de Greengrass incluyen añadir recursos de secretos al grupo, eliminar recursos de secretos del grupo y cambiar el conjunto de etiquetas de preparación que se incluyen en un recurso de secretos.

Si apunta a otro secreto de Secrets Manager, también debe editar los conectores que utilicen el secreto:

1. En la página de configuración del grupo, elija Connectors (Conectores).
2. En el menú contextual del conector, elija Edit (Editar).
3. La página Edit parameters (Editar parámetros) muestra un mensaje para informarle de que el ARN de secretos ha cambiado. Elija Guardar para confirmar el cambio.

Si elimina un secreto en Secrets Manager, elimine el recurso de secreto correspondiente del grupo y de los conectores y funciones de Lambda que hacen referencia a él. De lo contrario, durante la implementación del grupo, AWS IoT Greengrass devuelve un error que indica que no se puede encontrar el secreto. Actualice también su código de función de Lambda según sea necesario.

Creación y administración de recursos de secretos (CLI)

Creación de recursos de secretos (CLI)

En la API de AWS IoT Greengrass, un secreto es un tipo de recurso de grupo. En el siguiente ejemplo se crea una definición de recursos con una versión inicial que incluye un recurso de secretos llamado "`MySecretResource`". Para ver un tutorial que crea un recurso de secretos y lo añade a una versión de grupo, consulte the section called “Introducción a los conectores (CLI)”.

El recurso de secretos hace referencia al ARN del secreto de Secrets Manager correspondiente e incluye dos etiquetas de ensayo, además de `AWSCURRENT`, que siempre se incluye.

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
    "Resources": [
        {
            "Id": "my-resource-id",
            "Name": "MySecretResource",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
                    "AdditionalStagingLabelsToDownload": [
                        "Label1",
                        "Label2"
                    ]
                }
            }
```

```
        }
    ]
}'
```

Administración de recursos de secretos (CLI)

Las tareas de administración de los recursos de secretos de su grupo de Greengrass incluyen añadir recursos de secretos al grupo, eliminar recursos de secretos del grupo y cambiar el conjunto de etiquetas de preparación que se incluyen en un recurso de secretos.

En la API de AWS IoT Greengrass, estos cambios se implementan mediante la utilización de versiones.

La API de AWS IoT Greengrass usa versiones para administrar grupos. Las versiones son inmutables, por lo que para añadir o cambiar los componentes del grupo (por ejemplo, los dispositivos cliente, las funciones y los recursos del grupo), debe crear versiones de los componentes nuevos o actualizados. A continuación, cree y implemente una versión de grupo que contenga la versión de destino de cada componente. Para obtener más información acerca de los grupos, consulte the section called “AWS IoT Greengrass grupos”.

Por ejemplo, para cambiar el conjunto de etiquetas de ensayo de un recurso de secretos, siga estos pasos:

1. Cree una definición de recursos que contenga el recurso de secretos actualizado. En el siguiente ejemplo se añade una tercera etiqueta de ensayo el recurso de secretas de la sección anterior.

   > **Note**
   >
   > Para añadir más recursos a la versión, inclúyalos en la matriz `Resources`.

   ```
   aws greengrass create-resource-definition --name MyGreengrassResources --initial-
   version '{
       "Resources": [
           {
               "Id": "my-resource-id",
               "Name": "MySecretResource",
               "ResourceDataContainer": {
                   "SecretsManagerSecretResourceData": {
                       "ARN": "arn:aws:secretsmanager:us-
   west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
   ```

```
                    "AdditionalStagingLabelsToDownload": [
                        "Label1",
                        "Label2",
                        "Label3"
                    ]
                }
            }
        }
    ]
}'
```

2. Si el ID del recurso de secretos cambia, actualice los conectores y las funciones que utilizan el recurso de secretos. En las nuevas versiones, actualice el parámetro o la propiedad que se corresponde con el ID de recurso. Si el ARN del secreto cambia, también debe actualizar el parámetro correspondiente para los conectores que utilicen el secreto.

   > **Note**
   >
   > El ID de recurso es un identificador arbitrario que proporciona el cliente.

3. Crear una versión de grupo que contenga la versión de destino de cada componente que desea enviar al núcleo.
4. Implemente la versión de grupo.

Para ver un tutorial que muestre cómo crear e implementar recursos de secretos, conectores y funciones, consulte the section called “Introducción a los conectores (CLI)”.

Si elimina un secreto en Secrets Manager, elimine el recurso de secreto correspondiente del grupo y de los conectores y funciones de Lambda que hacen referencia a él. De lo contrario, durante la implementación del grupo, AWS IoT Greengrass devuelve un error que indica que no se puede encontrar el secreto. Actualice también su código de función de Lambda según sea necesario. Puede eliminar una secreto local mediante la implementación de una definición de recursos versión que no contenga el recurso de secretos correspondiente.

Uso de secretos locales en conectores y funciones de Lambda

Los conectores de Greengrass y las funciones de Lambda utilizan secretos locales para interactuar con servicios y aplicaciones. El valor `AWSCURRENT` se utiliza de forma predeterminada, pero los valores de otras etiquetas de ensayo incluido en el recurso de secretos también están disponibles.

Los conectores y las funciones deben configurarse antes de que puedan acceder a secretos locales. De este modo, el recurso de secretos se afilia con el conector o la función.

Connectors

Si un conector requiere acceso a un secreto local, proporciona parámetros que configura con la información que necesita para acceder al secreto.

- Para obtener información acerca de cómo hacerlo en la AWS IoT Greengrassconsola, consulte the section called “Introducción a los conectores (consola)”.
- Para obtener información acerca de cómo hacerlo en la CLI de AWS IoT Greengrass, consulte the section called “Introducción a los conectores (CLI)”.

Para obtener información acerca de los requisitos de conectores individuales, consulte the section called “conectores de Greengrass proporcionados por AWS”.

La lógica para acceder y utilizar el secreto está integrada en el conector.

Funciones de Lambda

Para permitir que una función de Lambda acceda a un secreto local, debe configurar las propiedades de la función.

- Para obtener información acerca de cómo hacerlo en la consola AWS IoT Greengrass, consulte the section called “Cómo crear un recurso de secreto (consola)”.
- Para hacer esto en la API de AWS IoT Greengrass, debe proporcionar la siguiente información en la propiedad `ResourceAccessPolicies`.
  - `ResourceId`: el ID del recurso de secretos en el grupo de Greengrass. Se trata del recurso que hace referencia al ARN del secreto de Secrets Manager correspondiente.
  - `Permission`: el tipo de acceso que la función tiene al recurso. Solo el permiso `ro` (solo lectura) es compatible con los recursos de secretos.

  En el siguiente ejemplo, se crea una función de Lambda que puede acceder al recurso de secretos `MyApiKey`.

```
aws greengrass create-function-definition --name MyGreengrassFunctions --initial-
version '{
    "Functions": [
        {
            "Id": "MyLambdaFunction",
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:1",
```

```
            "FunctionConfiguration": {
                "Pinned": false,
                "MemorySize": 16384,
                "Timeout": 10,
                "Environment": {
                    "ResourceAccessPolicies": [
                        {
                            "ResourceId": "MyApiKey",
                            "Permission": "ro"
                        }
                    ],
                    "AccessSysfs": true
                }
            }
        }
    ]
}'
```

Para acceder a secretos locales en tiempo de ejecución, las funciones de Lambda `get_secret_value` llaman a la función de Lambda `secretsmanager` desde el cliente de AWS IoT Greengrass en el Core SDK (versión 1.3.0 o posterior).

El siguiente ejemplo muestra cómo utilizar el Core SDK AWS IoT Greengrass para Python con el fin de obtener un secreto. Transfiere el nombre del secreto a la función `get_secret_value`. `SecretId` puede ser el nombre o ARN del secreto de Secrets Manager (no el recurso de secreto).

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-MySecret-abc"


def function_handler(event, context):
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret = response.get("SecretString")
```

Para secretos de tipo de texto, la función de Lambda `get_secret_value`devuelve una cadena. Para secretos de tipo binario, devuelve una cadena codificada en base64.

> **Important**
>
> Asegúrese de que las funciones de Lambda definidas por el usuario gestionen los secretos de forma segura y no registren ningún dato confidencial almacenado en el secreto. Para obtener más información, consulte Mitigar los riesgos de registrar y depurar la función de Lambda en la Guía del usuario de AWS Secrets Manager. Si bien esta documentación se refiere específicamente a las funciones de rotación, la recomendación también se aplica a las funciones de Lambda de Greengrass.

El valor actual del secreto se devuelve de forma predeterminada. Esta es la versión a la que está asociada la etiqueta de ensayo `AWSCURRENT`. Para acceder a una versión diferente, transfiera el nombre de la etiqueta de ensayo correspondiente para el argumento `VersionStage` opcional. Por ejemplo:

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-TestSecret"
secret_version = "MyTargetLabel"


# Get the value of a specific secret version
def function_handler(event, context):
    response = secrets_client.get_secret_value(
        SecretId=secret_name, VersionStage=secret_version
    )
    secret = response.get("SecretString")
```

Para otra función de ejemplo que llame a `get_secret_value`, consulte Creación de un paquete de implementación de la función de Lambda.

Cómo crear un recurso de secreto (consola)

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

Este tutorial muestra cómo utilizar la AWS Management Console para añadir un recurso de secreto a un grupo de Greengrass. Un recurso de secreto es una referencia a un secreto de AWS Secrets Manager. Para obtener más información, consulte Implementación de secretos en el núcleo.

En el dispositivo AWS IoT Greengrass principal, los conectores y las funciones de Lambda pueden usar el recurso de secretos para autenticarse con servicios y aplicaciones, sin necesidad de codificación rígida de contraseñas, tokens u otras credenciales.

En este tutorial, comience con la creación de un secreto en la consola de AWS Secrets Manager. A continuación, en la consola de AWS IoT Greengrass, añada un recurso secreto a un grupo de Greengrass de la página Recursos. Este recursos secreto hace referencia al secreto del Secrets Manager. Posteriormente, asocie un recurso secreto a una función de Lambda, que permite a la función obtener el valor del secreto local.

> **Note**
>
> Como alternativa, la consola le permite crear un secreto y un recurso secreto al configurar un conector o una función de Lambda. Puede hacerlo desde la página Configurar parámetros del conector o desde la página Recursos de la función de Lambda.
> Solo los conectores que contienen los parámetros de secretos pueden acceder a los secretos. Para obtener un tutorial que muestra la forma en que el conector de notificaciones Twilio utiliza un token de autenticación almacenado localmente, consulte the section called “Introducción a los conectores (consola)”.

El tutorial contiene los siguientes pasos generales:

1. Crear un secreto en Secrets Manager
2. Agregar un recurso de secreto a un grupo
3. Creación de un paquete de implementación de la función de Lambda
4. Creación de una función de Lambda
5. Agregar la función al grupo
6. Asociar el recurso de secreto a la función

7. Agregar suscripciones al grupo
8. Implementar el grupo
9. the section called “Probar la función de Lambda”

Completar el tutorial debería tomarle aproximadamente 20 minutos.

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Un grupo de Greengrass y un núcleo de Greengrass (versión 1.7 o posterior). Para obtener información acerca de cómo crear un núcleo y un grupo de Greengrass, consulte Empezando con AWS IoT Greengrass. El tutorial de introducción también incluye pasos para instalar el software AWS IoT Greengrass Core.
- AWS IoT Greengrass debe estar configurado para admitir secretos locales. Para obtener más información, consulte Requisitos de secretos.

  > **Note**
  >
  > Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Para obtener los valores de los secretos locales, las funciones de Lambda definidas por el usuario deben usar el SDK de AWS IoT Greengrass Core versión 1.3.0 o posterior.

Paso 1: Crear un secreto de Secrets Manager

En este paso, utilice la consola de AWS Secrets Manager para crear un secreto.

1. Inicie sesión en la consola de AWS Secrets Manager.

   > **Note**
   >
   > Para obtener más información sobre este proceso, consulte Paso 1: Crear y almacenar el secreto en AWS Secrets Manager en la Guía del usuario de AWS Secrets Manager.

2. Elija Store a new secret (Almacenar un nuevo secreto).
3. En Seleccionar tipo de secreto, elija Otro tipo de secretos.
4. En Specify the key-value pairs to be stored for this secret (Especificar los pares clave-valor que se almacenarán para este secreto):
   - En Key (Clave), escriba **`test`**.
   - En Value, ingrese **`abcdefghi`**.
5. Mantenga aws/secretsmanager seleccionado para la clave de cifrado y, a continuación, seleccione Siguiente.

   > **Note**
   >
   > AWS KMS no le cobrará si utiliza la clave administrada de AWS predeterminada que Secrets Manager crea en su cuenta.

6. En Secret name (Nombre del secreto), escriba **`greengrass-TestSecret`** y, a continuación, seleccione Next (Siguiente).

   > **Note**
   >
   > De forma predeterminada, el rol de servicio Greengrass permite a AWS IoT Greengrass obtener el valor de los secretos con nombres que comienzan por greengrass-. Para obtener más información, consulte requisitos de secretos.

7. Este tutorial no requiere rotación, así que elija desactivar la rotación automática y, a continuación, seleccione Siguiente.
8. En la página Review (Revisar), revise los ajustes y, a continuación, seleccione Store (Almacenar).

   A continuación, se crea un recurso de secreto en su grupo de Greengrass que hace referencia al secreto.

Paso 2: Agregar un recurso de secreto a un grupo de Greengrass

En este paso, debe configurar un recurso de grupo que hace referencia al secreto de Secrets Manager.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija el grupo al que desee añadir el recurso de secreto.
3. En la página de configuración del grupo, elija la pestaña Recursos y, a continuación, desplácese hacia abajo hasta la sección Secretos. La sección Secretos muestra los recursos secretos que pertenecen al grupo. Puede añadir, editar y quitar los recursos de secreto de esta sección.

   > **Note**
   >
   > Como alternativa, la consola le permite crear un secreto y un recurso secreto al configurar un conector o una función de Lambda. Puede hacerlo desde la página Configurar parámetros del conector o desde la página Recursos de la función de Lambda.

4. Seleccione Añadir en la sección Secretos.
5. En la página Añadir un recurso secreto, introduzca **`MyTestSecret`** el Nombre del recurso.
6. En Secreto, seleccione Greengrass-TestSecret.
7. En la sección Seleccionar etiquetas (opcional), la etiqueta provisional AWSCURRENT representa la versión más reciente del secreto. Esta etiqueta siempre está incluida en un recurso de secreto.

   > **Note**
   >
   > Este tutorial solo requiere la etiqueta AWSCURRENT. De forma opcional, puede incluir etiquetas que requieran su función de Lambda o un conector.

8. Seleccione Add resource (Añadir recurso).

Paso 3: Crear un paquete de implementación de la función de Lambda

Para crear una función de Lambda, primero debe crear un paquete de implementación de funciones de Lambda que contenga el código de la función y las dependencias. Las funciones de Lambda de Greengrass requieren el SDK de AWS IoT Greengrass Core para tareas como la comunicación con los mensajes de MQTT en el entorno principal y el acceso a los secretos locales. En este tutorial se crea una característica de Python para que utilices la versión Python del SDK en el paquete de implementación.

> **Note**
>
> Para obtener los valores de los secretos locales, las funciones de Lambda definidas por el usuario deben usar el SDK de AWS IoT Greengrass Core versión 1.3.0 o posterior.

1. Desde la página de descargas del Core SDK AWS IoT Greengrass, descargue el SDK AWS IoT Greengrass básico para Python en su ordenador.
2. Descomprima el paquete descargado para obtener el SDK. El SDK es la carpeta `greengrasssdk`.
3. Guarde la siguiente función de código de Python en un archivo local llamado "`secret_test.py`".

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
iot_client = greengrasssdk.client("iot-data")
secret_name = "greengrass-TestSecret"
send_topic = "secrets/output"


def function_handler(event, context):
    """
    Gets a secret and publishes a message to indicate whether the secret was
    successfully retrieved.
    """
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret_value = response.get("SecretString")
    message = (
        f"Failed to retrieve secret {secret_name}."
        if secret_value is None
        else f"Successfully retrieved secret {secret_name}."
    )
    iot_client.publish(topic=send_topic, payload=message)
    print("Published: " + message)
```

La función `get_secret_value` admite el nombre o ARN del secreto del Secrets Manager para el valor `SecretId`. En este ejemplo se utiliza el nombre del secreto. En este secreto de ejemplo, AWS IoT Greengrass devuelve el par clave-valor: `{"test":"abcdefghi"}`.

> **Important**
>
> Asegúrese de que las funciones de Lambda definidas por el usuario gestionen los secretos de forma segura y no registren ningún dato confidencial almacenado en el secreto. Para obtener más información, consulte Mitigar los riesgos de registrar y depurar la función de Lambda en la Guía del usuario de AWS Secrets Manager. Si bien esta documentación se refiere específicamente a las funciones de rotación, la recomendación también se aplica a las funciones de Lambda de Greengrass.

4. Comprima en un archivo ZIP los siguientes elementos en un archivo denominado "`secret_test_python.zip`". Al crear el archivo ZIP, incluya únicamente el código y sus dependencias, no la carpeta donde se encuentran.

   - secret_test.py. Lógica de la aplicación.
   - greengrasssdk. Biblioteca necesaria para todas las funciones de Lambda de Greengrass dd Python.

   Este es el paquete de implementación de la función de Lambda.

Paso 4: Crear una función de Lambda

En este paso, va a utilizar la consola de AWS Lambda para crear una función de Lambda y va a configurarla para utilizar su paquete de implementación. A continuación, publicará una versión de la función y creará un alias.

1. Primero, cree la función de Lambda.

   a. En la AWS Management Console, elija Services (Servicios) y abra la consola de AWS Lambda.

   b. Elija Crear función, y, a continuación, Autor desde cero.

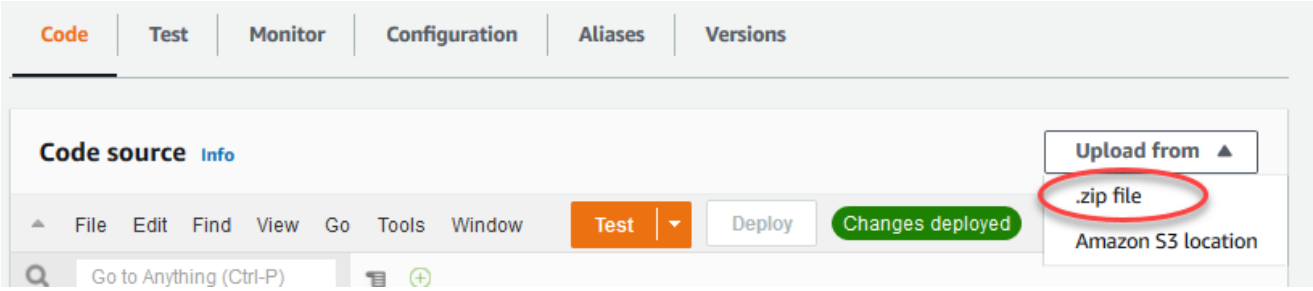
   c. En la sección Basic information (Información básica), utilice los siguientes valores:

- En Function name (Nombre de la función), introduzca **`SecretTest`**.
- En Runtime (Tiempo de ejecución), elija Python 3.7.
- En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol.

d. En la parte inferior de la página, elija Create function.

2. A continuación, registre el controlador y cargue el paquete de implementación de la función de Lambda.

a. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip.



b. Seleccione Cargar y, a continuación, elija su paquete de implementación `secret_test_python.zip`. A continuación, elija Save (Guardar).

c. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

- En Runtime (Tiempo de ejecución), elija Python 3.7.
- En Handler (Controlador), escriba **`secret_test.function_handler`**.

d. Seleccione Save.

> **Note**
>
> El botón de prueba de la consola de AWS Lambda no funciona con esta función. El SDK AWS IoT Greengrass Core no contiene los módulos necesarios para ejecutar las funciones de Lambda de Greengrass de forma independiente en la consola AWS Lambda. Estos módulos (por ejemplo, `greengrass_common`) se suministran a las funciones una vez desplegados en el núcleo de Greengrass.

3. Ahora, publique la primera versión de su función de Lambda y cree un alias para la versión.

> **Note**
>
> Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

a. En el menú Actions, elija Publish new version.

b. En Version description (Descripción de versión), escriba **`First version`** y, a continuación, elija Publish (Publicar).

c. En la página de configuración de SecretTest: 1, en el menú Actions (Acciones), elija Create alias (Crear alias).

d. En la página Create a new alias, utilice los valores siguientes:

- En Name (Nombre), ingrese **`GG_SecretTest`**.
- En Version (Versión), elija 1.

> **Note**
>
> AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST.

e. Seleccione Create (Crear).

Ahora está preparado para añadir la función de Lambda al grupo de Greengrass y asociar el recurso secreto.

Paso 5: Añadir la función de Lambda al grupo de Greengrass

En este paso, va a añadir la función de Lambda al grupo de Greengrass en la consola de AWS IoT.

1. En la página de configuración del grupo, elija la pestaña Funciones de lambda.
2. En la sección Mis funciones de Lambda, seleccione Añadir.
3. Para la función de Lambda, elija SecretTest.

4. Para la versión de la función de Lambda, elija el alias de la versión que publicó.

A continuación, configure el ciclo de vida de la función de Lambda.

1. En la sección de configuración de la función de Lambda, realice las siguientes actualizaciones.

> **Note**
>
> Le recomendamos que ejecute la función de Lambda sin creación de contenedores, a menos que su modelo de negocio lo requiera. Esto permite el acceso a la GPU y la cámara del dispositivo sin necesidad de configurar los recursos del dispositivo. Si ejecuta sin creación de contenedores, también debe conceder acceso raíz a las funciones de Lambda AWS IoT Greengrass.

   a. Para ejecutar sin creación de contenedores:

      - En Usuario y grupo del sistema, elija **`Another user ID/group ID`**. En ID de usuario del sistema, introduzca **`0`**. Para el ID de grupo del sistema, introduzca **`0`**.

        Esto permite que la función de Lambda se ejecute de raíz. Para obtener más información sobre cómo ejecutar como raíz, consulte the section called “Configuración de la identidad de acceso predeterminada para las funciones de Lambda de un grupo”.

        > **Tip**
        >
        > También debe actualizar el archivo `config.json` para conceder acceso raíz a la función de Lambda. Para el procedimiento, consulte the section called “Ejecución de una función de Lambda como raíz”.

      - Para la creación de contenedores de la función de Lambda, elija Sin contenedor.

        Para obtener más información sobre la ejecución sin creación de contenedores, consulte the section called “Consideraciones a la hora de elegir la creación de contenedores de la función de Lambda.”.

      - En Tiempo de espera, escriba **`10 seconds`**.
      - En Ancladas, elija Verdadero

Para obtener más información, consulte the section called “Configuración del ciclo de vida”.

- En Parámetro adicional, para Acceso de lectura al directorio /sys, elija Activado.

b. Para ejecutarlo en modo contenerizado, en su lugar:

> **Note**
>
> No recomendamos ejecutarlo en modo contenerizado a menos que su modelo de negocio lo requiera.

- En Usuario y grupo del sistema, seleccione Usar grupo predeterminado.
- Para la creación de contenedores de funciones de Lambda, elija Usar grupo por defecto.
- En Límite de memoria, escriba **`1024 MB`**.
- En Tiempo de espera, escriba **`10 seconds`**.
- En Ancladas, elija Verdadero

  Para obtener más información, consulte the section called “Configuración del ciclo de vida”.
- En Parámetros adicionales, para Acceso de lectura al directorio /sys, elija Activado.

2. Elija Añadir función de Lambda.

A continuación, el recurso de secretos se aocias con la función.

Paso 6: Asociar el recurso de secreto a la función de Lambda

En este paso, asocie el recurso secreto a la función de Lambda en su grupo de Greengrass. Esto asocia el recurso con la función, que permite a la función obtener el valor del secreto local.

1. En la página de configuración del grupo, elija la pestaña Funciones de lambda.
2. Seleccione la función SecretTest.
3. En la página de detalles de la función, seleccione Recursos.
4. Vaya a la sección Secretos y seleccione Asociar.
5. Elija MyTestSecret y, a continuación, seleccione Asociar.

Paso 7: Agregar suscripciones al grupo de Greengrass

En este paso, añada las suscripciones que permitan a AWS IoT y la función de Lambda intercambiar mensajes. Una suscripción permite a AWS IoT invocar la función y un permite a la función para enviar datos de salida a AWS IoT.

1. En la página de configuración del grupo, elija la pestaña Suscripciones y, a continuación, elija Añadir suscripción.
2. Cree crear una suscripción que permita a AWS IoT publicar mensajes en la función.

   En la página de configuración del grupo, elija la pestaña Suscripciones y, a continuación, elija Añadir suscripción.
3. En la página Crear una suscripción, configure el origen y el destino de la siguiente manera:

   a. En Tipo de origen, elija función de Lambda y, a continuación, elija Nube IoT.
   b. En Tipo de destino, elija Servicio y, a continuación, SecretTest.
   c. En Filtro por temas, introduzca **`secrets/input`** y, a continuación, seleccione Crear suscripción.
4. Añadir una segunda suscripción. Seleccione la pestaña Suscripciones, elija Agregar suscripción y configure el origen y el destino de la siguiente manera:

   a. En Tipo de fuente, elija Servicios y, a continuación, SecretTest.
   b. En Tipo de destino, elija función de Lambda y, a continuación, elija Nube IoT.
   c. En Filtro por temas, introduzca **`secrets/output`** y, a continuación, seleccione Crear suscripción.

Paso 8: Implementar el grupo de Greengrass

Implemente el grupo en el dispositivo del núcleo. Durante la implementación, AWS IoT Greengrass recupera el valor del secreto de Secrets Manager y crea una copia cifrada y local en el núcleo.

1. Asegúrese de que el núcleo de AWS IoT Greengrass se está ejecutando. Ejecute los siguientes comandos en el terminal de Raspberry Pi según sea necesario.

   a. Para comprobar si el demonio está en ejecución:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, el demonio está en ejecución.

> **Note**
>
> La versión que figura en la ruta depende de la versión del software AWS IoT Greengrass Core que esté instalada en el dispositivo del núcleo.

b. Iniciar el daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. En la página de configuración de grupo, elija Implementar.
3. a. En la pestaña Funciones de Lambda, en la sección Funciones de Lambda del sistema, seleccione Detector de IP y elija Editar.

   b. En el cuadro de diálogo Editar configuración del detector IP, seleccione Detectar y anular automáticamente los puntos de conexión del agente MQTT.

   c. Seleccione Save.

   Esto permite a los dispositivos adquirir automáticamente la información de conexión del dispositivo principal, como la dirección IP, el DNS y el número de puerto. Se recomienda la detección automática, pero AWS IoT Greengrass también es compatible con puntos de enlace especificados manualmente. Solo se le solicitará el método de detección la primera vez que se implemente el grupo.

   > **Note**
   >
   > Si se le solicita, conceda permiso para crear el rol de servicio de Greengrass y asócielo a su Cuenta de AWS en el Región de AWS actual. Este rol permite a AWS IoT Greengrass acceder a los servicios de AWS.

   En la página Deployments (Implementaciones), se muestra la marca temporal, el ID de versión y el estado de la implementación. Una vez terminada, la implementación debería mostrar el estado Completado.

Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

Probar la función de Lambda

1. En la página de inicio de la consola AWS IoT, elija Pruebas.
2. Para Suscribirse al tema, utilice los siguientes valores y, a continuación, seleccione Suscribirse.

Propiedad	Valor
Subscription topic	secrets/output
Visualización de la carga de MQTT	Display payloads as strings

3. Para Publicar en tema, utilice los siguientes valores y, a continuación, seleccione Publicar para invocar la función.

Propiedad	Valor
Tema	secrets/input
Mensaje	Conserve el mensaje predeterminado. La publicación de un mensaje invoca la función de Lambda, pero la función de este tutorial no procesa el cuerpo del mensaje.

Si se ejecuta correctamente, la función publica un mensaje de "Success (Correcto)".

Véase también

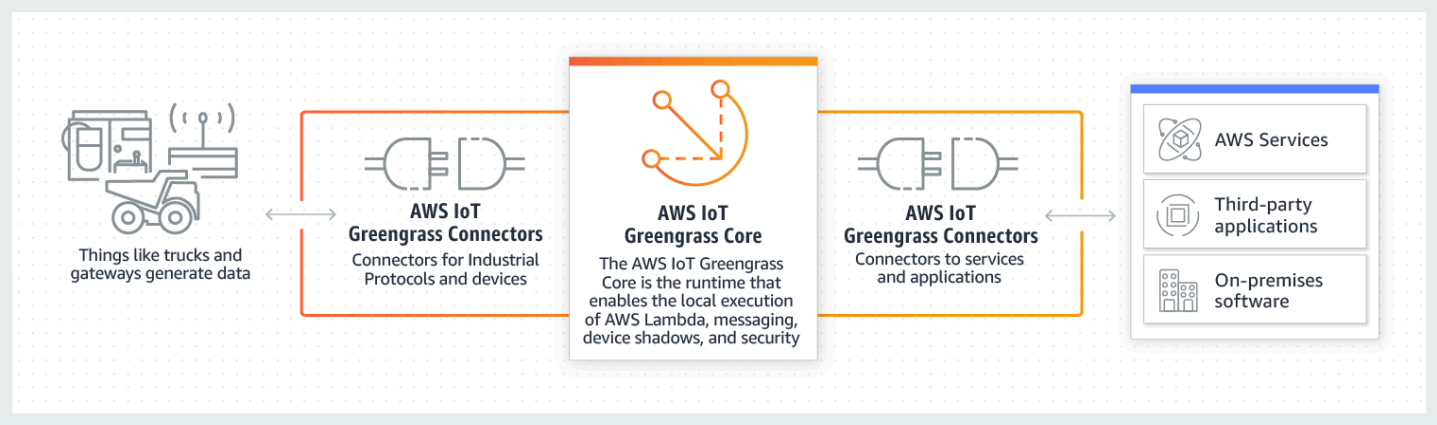
- Implementación de secretos en el núcleo

Integración con servicios y protocolos mediante conectores de Greengrass

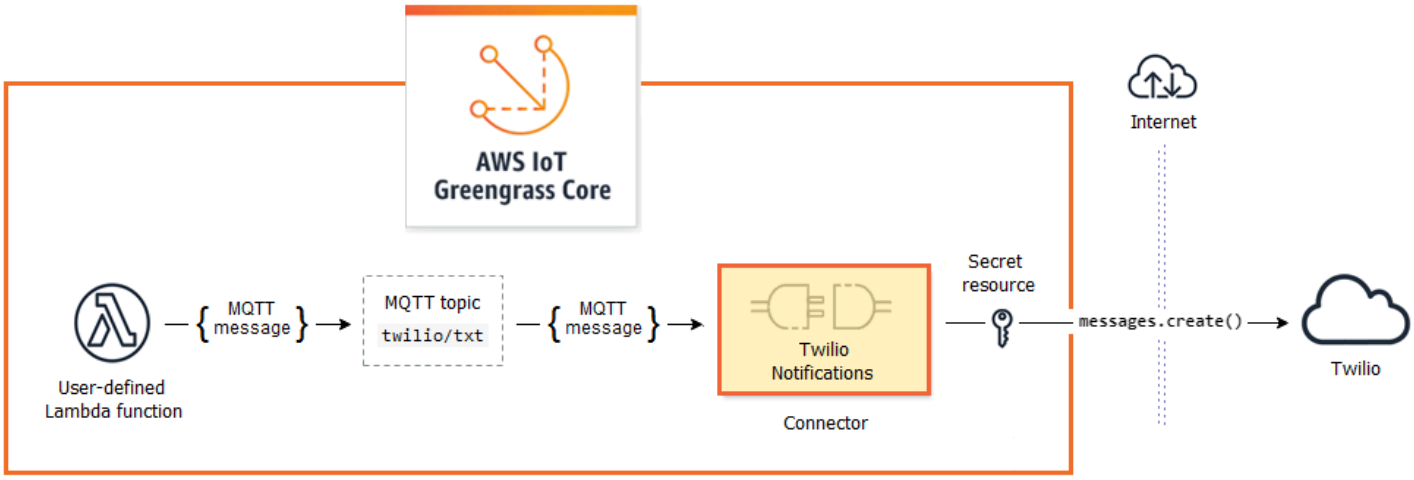
Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y versiones posteriores.

Los conectores en AWS IoT Greengrass son módulos preconstruidos que hacen más eficaz la interacción con la infraestructura local, los protocolos de los dispositivos, AWS, y otros servicios en la nube. Usando conectores le permitirá dedicar menos tiempo a aprender nuevos protocolos e interfaces API, y más tiempo a centrarse en la lógica importante para su negocio.

El siguiente diagrama muestra dónde encajan los conectores en el entorno de AWS IoT Greengrass .



Muchos conectores utilizan mensajes de MQTT para comunicarse con dispositivos y funciones de Lambda del grupo o con AWS IoT y el servicio de sombra local. En el siguiente ejemplo, el conector de notificaciones de Twilio recibe mensajes MQTT de una función de Lambda definida por el usuario, utiliza una referencia local de un secreto de AWS Secrets Manager, y llama a la API de Twilio.



Para ver tutoriales que crean esta solución, consulte the section called “Introducción a los conectores (consola)” y the section called “Introducción a los conectores (CLI)”.

Los conectores de Greengrass pueden ayudarle a ampliar las posibilidades del dispositivo o crear dispositivos de una sola finalidad. Mediante el uso de conectores, puede:

- Implementar una lógica de negocio reutilizable.
- Interactuar con servicios locales y de nube, incluidos los servicios de AWS y de terceros.
- Adquirir y procesar los datos del dispositivo.
- Habilitar las llamadas de un dispositivo a otro con suscripciones de temas de MQTT y funciones de Lambda definidas por el usuario.

AWS ofrece un conjunto de conectores de Greengrass que simplifican las interacciones con los servicios comunes y los orígenes de datos. Estos módulos preconfigurados, permiten escenarios para el registro y diagnóstico, reabastecimiento, procesamiento de datos industriales, alarma y mensajería. Para obtener más información, consulte the section called “conectores de Greengrass proporcionados por AWS”.

Requisitos

Para usar conectores, tenga en cuenta lo siguiente:

- Cada conector que utilice tiene unos requisitos que debe cumplir. Estos requisitos pueden incluir la versión mínima del software AWS IoT Greengrass Core, los requisitos previos del dispositivo,

los permisos necesarios y los límites. Para obtener más información, consulte the section called “conectores de Greengrass proporcionados por AWS”.

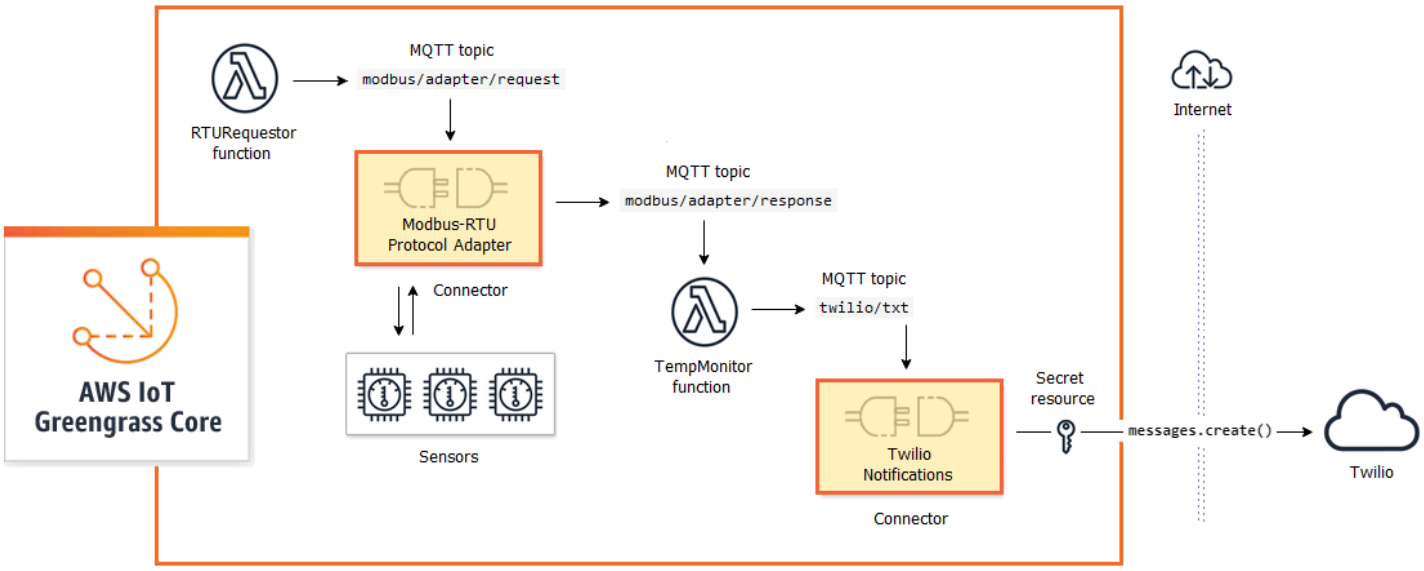
- Un grupo Greengrass sólo puede contener una instancia configurada de un conector determinado. Sin embargo, puede utilizar la instancia en varias suscripciones. Para obtener más información, consulte the section called “Parámetros de configuración”.
- Cuando la creación de contenedores predeterminada para el grupo de Greengrass se establece en Sin contenedor, los conectores del grupo deben ejecutarse sin creación de contenedores. Para buscar conectores compatibles con el modo Sin contenedor, consulte the section called “conectores de Greengrass proporcionados por AWS”.

Uso de conectores de Greengrass

Un conector es un tipo de componente del grupo. Al igual que otros componentes del grupo, como dispositivos cliente y funciones de Lambda definidas por el usuario, puede agregar conectores a los grupos, configurarlos e implementarlos en el nucleo AWS IoT Greengrass. Los conectores se ejecutan en el entorno de núcleo.

Algunos conectores pueden implementarse como aplicaciones independientes simples. Por ejemplo, el conector Device Defender lee las métricas del sistema del dispositivo central y las envía a AWS IoT Device Defender para su análisis.

Puede añadir otros conectores como bloques de construcción en soluciones más amplias. La siguiente solución de ejemplo utiliza el conector Adaptador de protocolo Modbus-RTU para procesar los mensajes de los sensores y el conector Notificaciones Twilio para iniciar los mensajes Twilio.



Las soluciones suelen incluir funciones de Lambda definidas por el usuario que se colocan junto a los conectores y procesan los datos que el conector envía o recibe. En este ejemplo, la función TempMonitor recibe datos del adaptador de protocolo Modbus-RTU, ejecuta cierta lógica empresarial y, a continuación, envía datos a Twilio Notifications.

Para crear e implementar una solución, debe seguir este proceso general:

1. Planificar el flujo de datos de alto nivel. Identificar los orígenes de datos, los canales de datos, los servicios, los protocolos y los recursos con los que necesita trabajar. En la solución de ejemplo, esto incluye los datos a través del protocolo Modbus RTU, el puerto de serie Modbus físico y Twilio.
2. Identificar los conectores que se incluirán en la solución y añadirlos a su grupo. La solución de ejemplo utiliza el adaptador de protocolo Modbus-RTU y las notificaciones de Twilio. Para ayudarle a detectar los conectores aplicables a su situación y para ver más información acerca de sus requisitos individuales, consulte the section called “conectores de Greengrass proporcionados por AWS”.
3. Identificar si las funciones de Lambda definidas por el usuario, dispositivos cliente o recursos son necesarios y, a continuación, crearlas y añadirlas al grupo. Esto podría incluir funciones que contienen lógica de negocio o procesar datos en un formato requerido por otra entidad en la solución. La solución de ejemplo utiliza funciones para enviar las solicitudes Modbus RTU e iniciar notificaciones Twilio. También incluye un recurso de dispositivo local para el puerto de serie Modbus RTU y un recurso de secreto para el token de autenticación de Twilio.

> **Note**
>
> Los recursos de secreto hacen referencia a contraseñas, tokens y otros secretos de AWS Secrets Manager. Los conectores y las funciones de Lambda pueden usar secretos para autenticar con servicios y aplicaciones. De forma predeterminada, AWS IoT Greengrass puede obtener acceso a los secretos cuyos nombres empiezan por greengrass-. Para obtener más información, consulte Implementación de secretos en el núcleo.

4. Crear las suscripciones que permiten que las entidades de la solución intercambien mensajes MQTT. Si un conector se utiliza en una suscripción, el conector y el origen o destino del mensaje deben utilizar la sintaxis de tema predefinida compatible con el conector. Para obtener más información, consulte the section called “Entradas y salidas”.
5. Implementar el grupo en el núcleo de Greengrass.

Para obtener información acerca de cómo crear e implementar un conector, consulte los siguientes tutoriales:

- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”

Parámetros de configuración

Muchos conectores proporcionan parámetros que permiten personalizar su comportamiento o su salida. Estos parámetros se utilizan durante la inicialización, el tiempo de ejecución o en otros momentos en el ciclo de vida del conector.

Los tipos de parámetros y su uso varían según el conector. Por ejemplo, el conector SNS tiene un parámetro que configura el tema SNS por defecto, y Device Defender tiene un parámetro que configura la frecuencia de muestreo de datos.

Una versión de grupo puede contener varios conectores, pero solo una instancia de un determinado conector a la vez. Esto significa que cada conector del grupo solamente puede tener una configuración activa. Sin embargo, la instancia del conector puede utilizarse en diferentes suscripciones del grupo. Por ejemplo, puede crear suscripciones que permitan a muchos dispositivos enviar datos al conector Kinesis Firehose.

Parámetros utilizados para acceder a recursos de grupo

Los conectores de Greengrass utilizan recursos de grupo para obtener acceso al sistema de archivos, puertos, periféricos y otros recursos locales en el dispositivo principal. Si un conector requiere acceso a un recurso de grupo, proporciona parámetros de configuración relacionados.

Los recursos de grupo incluyen:

- Recursos locales. Los directorios, archivos, puertos, pines y periféricos que están presentes en el dispositivo del núcleo de Greengrass.
- Recursos de machine learning. Los modelos de machine learning que se utilizan en la nube y se implementan en el núcleo para la inferencia local.
- Recurso de secreto. Locales, copias cifradas de contraseñas, claves, tokens o texto arbitrario de AWS Secrets Manager. Los conectores pueden acceder de manera segura a estos secretos locales y utilizarlos para autenticarse en los servicios o en la infraestructura local.

Por ejemplo, los parámetros para Device Defender permiten acceder a las métricas del sistema en el directorio `/proc` del host, y los parámetros para Twilio Notifications permiten acceder a un token de autenticación Twilio almacenado localmente.

Actualización de parámetros de conectores

Los parámetros se configuran cuando el conector se añade a un grupo de Greengrass. Puede cambiar los valores de los parámetros después de añadir un conector.

- En la consola: en la página de configuración de grupo, abra Connectors (Conectores) y desde el menú contextual del conector, seleccione Edit (Editar).

  > **Note**
  >
  > Si el conector utiliza un recurso de secreto que más adelante cambia para hacer referencia a otro secreto, debe editar los parámetros del conector y confirmar el cambio.

- En la API: cree otra versión del conector que defina la nueva configuración.

  La API de AWS IoT Greengrass usa versiones para administrar grupos. Las versiones son inmutables, por lo que para añadir o cambiar los componentes del grupo (por ejemplo, los dispositivos cliente, las funciones y los recursos del grupo), debe crear versiones de los

componentes nuevos o actualizados. A continuación, cree e implemente una versión de grupo que contenga la versión de destino de cada componente.

Después de realizar cambios en la configuración del conector, debe implementar el grupo para propagar los cambios en el núcleo.

Entradas y salidas

Muchos conectores de Greengrass pueden comunicarse con otras entidades enviando y recibiendo mensajes MQTT. Esta comunicación MQTT se controla mediante suscripciones que permiten a un conector intercambiar datos con las funciones de Lambda, los dispositivos y otros conectores dentro del grupo de Greengrass, o con AWS IoT y el servicio de sombra local. Para permitir esta comunicación, debe crear suscripciones en el grupo al que pertenece el conector. Para obtener más información, consulte the section called “Suscripciones administradas en el flujo de trabajo de mensajería de MQTT”.

Los conectores pueden ser suscriptores de mensajes, publicadores de mensajes o ambos. Cada conector define los temas de MQTT a los que se suscribe o en los que publica. Estos temas predefinidos deben utilizarse en las suscripciones en las que el conector sea un origen de mensaje o destino de mensaje. Para ver tutoriales con pasos para configurar suscripciones para un conector, consulte the section called “Introducción a los conectores (consola)” y the section called “Introducción a los conectores (CLI)”.

> **Note**
>
> Muchos conectores también disponen de modos integrados de comunicación para interactuar con servicios locales o en la nube. Estos varían según el conector y podrían requerir configurar parámetros o agregar permisos al rol de grupo. Para obtener información sobre los requisitos de los conectores, consulte the section called “conectores de Greengrass proporcionados por AWS”.

Temas de entrada

La mayoría de los conectores reciben datos de entrada sobre temas de MQTT. Algunos conectores se suscriben a varios temas para los datos de entrada. Por ejemplo, el conector Serial Stream admite dos temas:

- `serial/+/read/#`
- `serial/+/write/#`

Para este conector, las solicitudes de lectura y escritura se envían al tema correspondiente. Al crear las suscripciones, asegúrese de que se va a utilizar el tema que se adapte a su implementación.

Los caracteres + y # en los ejemplos anteriores son comodines. Estos comodines permiten que los suscriptores reciban mensajes sobre varios temas y que los editores personalicen los temas que publican.

- El comodín + puede aparecer en cualquier lugar de la jerarquía de temas. Se puede sustituir por un elemento de la jerarquía.

  Por ejemplo, en el caso del tema `sensor/+/input`, los mensajes se pueden publicar en los temas `sensor/id-123/input`, pero no en `sensor/group-a/id-123/input`.

- El comodín # puede aparecer solo al final de la jerarquía de temas. Se puede sustituir por cualquier número de elementos de la jerarquía, incluso ninguno.

  Por ejemplo, en el caso del tema `sensor/#`, los mensajes se pueden publicar en `sensor/`, `sensor/id-123` y `sensor/group-a/id-123`, pero no en `sensor`.

Los caracteres comodín solo son válidos al suscribirse a temas. Los mensajes no se pueden publicar en temas que contengan comodines. Consulte la documentación del conector para obtener más información sobre los requisitos de su tema de entrada o salida. Para obtener más información, consulte the section called “conectores de Greengrass proporcionados por AWS”.

Soporte de creación de contenedores

De forma predeterminada, la mayoría de los conectores se ejecutan en el núcleo de Greengrass en un entorno de tiempo de ejecución aislado administrado por AWS IoT Greengrass. Estos entornos de tiempo de ejecución, denominados contenedores, proporcionan aislamiento entre los conectores y el sistema del host, lo que ofrece más seguridad para el host y el conector.

Sin embargo, esta creación de contenedores de Greengrass no es compatible en algunos entornos, como la ejecución de AWS IoT Greengrass en un contenedor de Docker o en núcleos Linux antiguos sin cgroups. En estos entornos, los conectores deben ejecutarse en modo Sin contenedor . Para

buscar conectores compatibles con el modo Sin contenedor, consulte the section called “conectores de Greengrass proporcionados por AWS”. Algunos conectores se ejecutan en este modo de forma nativa, y algunos conectores le permiten establecer el modo de aislamiento.

También puede establecer el modo de aislamiento en Sin contenedor en entornos que admiten la creación de contenedores de Greengrass, pero se recomienda utilizar el modo contenedor Greengrass cuando sea posible.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo Greengrass no se aplica a los conectores.

Actualización de versiones de los conectores

Los proveedores de conectores pueden lanzar nuevas versiones de un conector que agregan características, solucionan problemas o mejoran el rendimiento. Para obtener información sobre las versiones disponibles y los cambios relacionados, consulte la documentación de cada conector.

En la consola AWS IoT, puede buscar nuevas versiones para los conectores de su grupo Greengrass.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. En Grupos de Greengrass, elija su grupo.
3. Elija Connectors (Conectores) para mostrar los conectores del grupo.

   Si el conector tiene una nueva versión, aparecerá un botón Available (Disponible) en la columna Upgrade (Actualizar).
4. Para actualizar la versión del conector:

   a. En la página Connectors (Conectores) en la columna Upgrade (Actualizar) seleccione Available (Disponible). Se abre la página Upgrade connector (Actualizar conector) y muestra los parámetros actuales, cuando corresponda.

      Elija la nueva versión del conector, defina los parámetros según sea necesario y, a continuación, elija Upgrade (Actualizar).

b. En la página Subscriptions (Suscripciones) agregue nuevas suscripciones al grupo para reemplazar las que utilicen el conector como origen o destino. A continuación, elimine las suscripciones antiguas.

   Las suscripciones hacen referencia a conectores por versión, por lo que no son válidos si cambia la versión del conector en el grupo.

c. En el menú Actions (Acciones) elija Deploy (Implementar) para implementar los cambios en el núcleo.

Para actualizar un conector desde la API de AWS IoT Greengrass, cree e implemente una versión de grupo que incluya el conector actualizado y las suscripciones. Utilice el mismo proceso que al agregar un conector a un grupo. Para ver los pasos detallados que le muestran cómo utilizar el AWS CLI para configurar y desplegar un conector de Twilio Notifications de ejemplo, consulte the section called “Introducción a los conectores (CLI)”.

Registro de conectores

Los conectores de Greengrass contienen funciones de Lambda que escriben eventos y errores en los registros de Greengrass. En función de su configuración del grupo, los registros se escriben en CloudWatch Logs, el sistema de archivos local o ambos. Los registros de los conectores incluyen el ARN de la característica correspondiente. El siguiente ejemplo de ARN procede del conector Kinesis Firehose:

```
arn:aws:lambda:aws-region:account-id:function:KinesisFirehoseClient:1
```

La configuración de registro predeterminada escribe los registros de nivel de información en el sistema de archivos mediante la siguiente estructura de directorios:

```
greengrass-root/ggc/var/log/user/region/aws/function-name.log
```

Para obtener más información sobre los registros de Greengrass, consulte the section called “Monitorización con registros de AWS IoT Greengrass”.

conectores de Greengrass proporcionados por AWS

AWS proporciona los siguientes conectores que admiten AWS IoT Greengrass situaciones comunes. Para obtener más información acerca de cómo funcionan los conectores, consulte la siguiente documentación:

- Integración con servicios y protocolos mediante conectores
- Introducción a los conectores (consola) o Introducción a los conectores (CLI)

Kinesis - S3	Descripción	Tiempos de ejecución compatibles de Lambda	Admite el modo Sin contenedor
CloudWatch Métricas	Publica métricas personalizadas en Amazon CloudWatch.	• Python 3.8 * • Python 3.7 • Python 2.7	Sí
Device Defender	Envía las métricas del sistema a AWS IoT Device Defender.	• Python 3.8 * • Python 3.7 • Python 2.7	No
Implementación de aplicación Docker	Ejecuta un archivo de Docker Compose para iniciar una aplicación Docker en el dispositivo central.	• Python 3.8 • Python 3.7	Sí
IoT Analytics	Envía datos desde dispositivos y sensores a AWS IoT Analytics.	• Python 3.8 * • Python 3.7 • Python 2.7	Sí

Kinesis - S3	Descripción	Tiempos de ejecución compatibles de Lambda	Admite el modo Sin contenedor
Adaptador de protocolo IP Ethernet IoT	Recopila datos de dispositivos EtherNet/IP.	• Java 8	Sí
IoT SiteWise	Envía datos desde dispositivos y sensores a propiedades de recurso en AWS IoT SiteWise.	• Java 8	Sí
Kinesis Firehose	Envía datos a las transmisiones de entrega de Amazon Data Firehose.	• Python 3.8 * • Python 3.7 • Python 2.7	Sí
ML Feedback	Publica la entrada del modelo de machine learning en la nube y la salida en un tema de MQTT.	• Python 3.8 • Python 3.7	No
Clasificación de imágenes de ML	Ejecuta un servicio de inferencia de clasificación de una imagen local. Este conector proporciona versiones para varias plataformas.	• Python 3.8 * • Python 3.7 • Python 2.7	No
Detección de objetos de ML	Ejecuta un servicio de inferencia de detección de objetos local. Este conector proporciona versiones para varias plataformas.	• Python 3.8 • Python 3.7	No
Adaptador de protocolo Modbus-RTU	Envía solicitudes a dispositivos Modbus RTU.	• Python 3.8 * • Python 3.7 • Python 2.7	No

Kinesis - S3	Descripción	Tiempos de ejecución compatibles de Lambda	Admite el modo Sin contenedor
Adaptador de protocolo Modbus-TCP	Recopila datos de los dispositivos ModbusTCP.	• Java 8	Sí
Raspberry Pi GPIO	Controla los pines de GPIO en un dispositivo de núcleo Raspberry Pi.	• Python 3.8 * • Python 3.7 • Python 2.7	No
Serial Stream	Las operaciones de lectura y escritura en un puerto de serie del dispositivo de núcleo.	• Python 3.8 * • Python 3.7 • Python 2.7	No
ServiceNow MetricBase Integración	Publica métricas de series temporales para ServiceNow MetricBase.	• Python 3.8 * • Python 3.7 • Python 2.7	Sí
SNS	Envía mensajes a un tema de Amazon SNS.	• Python 3.8 * • Python 3.7 • Python 2.7	Sí
Integración de Splunk	Publica datos en Splunk HEC.	• Python 3.8 * • Python 3.7 • Python 2.7	Sí

Kinesis - S3	Descripción	Tiempos de ejecución compatibles de Lambda	Admite el modo Sin contenedor
Notificaciones de Twilio	Inicia un mensaje de texto o voz de Twilio.	• Python 3.8 * • Python 3.7 • Python 2.7	Sí

* Para utilizar los tiempos de ejecución de Python 3.8, debe crear un enlace simbólico desde la carpeta de instalación por defecto de Python 3.7 a los binarios instalados de Python 3.8. Para obtener más información, consulte los requisitos específicos del conector.

> **Note**
>
> Recomendamos que actualice las versiones de conector de Python 2.7 a Python 3.7. El soporte continuo para los conectores de Python 2.7 depende del soporte en AWS Lambda tiempo de ejecución. Para obtener más información, consulte la Política de soporte en tiempo de ejecución en la Guía de desarrolladores de AWS Lambda .

CloudWatch Conector de métricas

El conector CloudWatch Metrics publica métricas personalizadas de los dispositivos Greengrass en Amazon. CloudWatch El conector proporciona una infraestructura centralizada para publicar CloudWatch métricas, que puede utilizar para supervisar y analizar el entorno principal de Greengrass y actuar en función de los eventos locales. Para obtener más información, consulta Uso de CloudWatch las métricas de Amazon en la Guía del CloudWatch usuario de Amazon.

Este conector recibe datos de métricas como mensajes de MQTT. El conector agrupa las métricas que se encuentran en el mismo espacio de nombres y las publica a CloudWatch intervalos regulares.

Este conector tiene las siguientes versiones.

Versión	ARN
5	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/5`
4	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/4`
3	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/3`
2.	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/2`
1	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3 - 5

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

> **Note**
>
> Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- El rol de grupo Greengrass configurado para permitir la acción `cloudwatch:PutMetricData`, tal como se muestra en la política de (IAM) AWS Identity and Access Management de ejemplo siguiente.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Para obtener más información sobre CloudWatch los permisos, consulta la referencia de CloudWatch permisos de Amazon en la Guía del usuario de IAM.

Versions 1 - 2

- Software AWS IoT Greengrass Core versión 1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- El rol de grupo Greengrass configurado para permitir la acción `cloudwatch:PutMetricData`, tal como se muestra en la política de (IAM) AWS Identity and Access Management de ejemplo siguiente.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Para obtener más información sobre CloudWatch los permisos, consulta la referencia de CloudWatch permisos de Amazon en la Guía del usuario de IAM.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

Versions 4 - 5

`PublishInterval`

El número máximo de segundos que se debe esperar antes de publicar las métricas por lote para un determinado espacio de nombres. El valor máximo es 900. Para configurar el conector con el fin de publicar las métricas a medida que se reciben (sin hacerlo por lotes), especifique 0.

El conector publica para CloudWatch después de recibir 20 métricas en el mismo espacio de nombres o después del intervalo especificado.

> **Note**
>
> El conector no garantiza el orden de publicación de los eventos.

Nombre para mostrar en la consola AWS IoT: Publicar intervalo

Obligatorio: `true`

Tipo: `string`

Valores válidos: `0 - 900`

Patrón válido: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

El Región de AWS para publicar CloudWatch las métricas. Este valor anula la región de métricas de Greengrass predeterminada. Solo se requiere cuando se publican métricas entre regiones.

Nombre para mostrar en la consola AWS IoT: Publicar región

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

La memoria (en KB) que se asignará al conector.

Nombre que mostrar en la consola AWS IoT: Tamaño de memoria

Obligatorio: `true`

Tipo: `string`

Patrón válido: `^[0-9]+$`

`MaxMetricsToRetain`

El número máximo de métricas de todos los espacios de nombres que se guardarán en la memoria antes de que se reemplacen por nuevas métricas. El valor mínimo es 2000.

Este límite se aplica cuando no hay conexión a Internet y el conector comienza a almacenar en búfer las métricas que se van a publicar más adelante. Cuando el búfer está lleno, la métricas más antiguas se sustituyen por nuevas. Las métricas de un determinado espacio de nombres se reemplazan únicamente por métricas del mismo espacio de nombres.

> **Note**
>
> Las métricas no se guardan si el proceso de host para el conector se interrumpe. Por ejemplo, esta interrupción puede ocurrir durante la implementación de grupos o cuando el dispositivo se reinicia.

Nombre para mostrar en la consola AWS IoT: Número máximo de métricas que se deben retener

Obligatorio: `true`

Tipo: `string`

Patrón válido: `^([2-9]\d{3}|[1-9]\d{4,})$`

`IsolationMode`

El modo de creación de contenedores para este conector. El valor predeterminado es `GreengrassContainer`, lo que significa que el conector se ejecuta en un entorno de tiempo de ejecución aislado dentro del contenedor de AWS IoT Greengrass.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo no se aplica a los conectores.

Nombre para mostrar en la consola AWS IoT: Modo de aislamiento de contenedores

Obligatorio: `false`

Tipo: `string`

Valores válidos: `GreengrassContainer` o `NoContainer`

Patrón válido: `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`PublishInterval`

El número máximo de segundos que se debe esperar antes de publicar las métricas por lote para un determinado espacio de nombres. El valor máximo es 900. Para configurar el conector con el fin de publicar las métricas a medida que se reciben (sin hacerlo por lotes), especifique 0.

El conector publica para CloudWatch después de recibir 20 métricas en el mismo espacio de nombres o después del intervalo especificado.

> **Note**
>
> El conector no garantiza el orden de publicación de los eventos.

Nombre para mostrar en la consola AWS IoT: Publicar intervalo

Obligatorio: `true`

Tipo: `string`

Valores válidos: `0 - 900`

Patrón válido: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

El Región de AWS para publicar CloudWatch las métricas. Este valor anula la región de métricas de Greengrass predeterminada. Solo se requiere cuando se publican métricas entre regiones.

Nombre para mostrar en la consola AWS IoT: Publicar región

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

La memoria (en KB) que se asignará al conector.

Nombre que mostrar en la consola AWS IoT: Tamaño de memoria

Obligatorio: `true`

Tipo: `string`

Patrón válido: `^[0-9]+$`

`MaxMetricsToRetain`

El número máximo de métricas de todos los espacios de nombres que se guardarán en la memoria antes de que se reemplacen por nuevas métricas. El valor mínimo es 2000.

Este límite se aplica cuando no hay conexión a Internet y el conector comienza a almacenar en búfer las métricas que se van a publicar más adelante. Cuando el búfer está lleno, la métricas más antiguas se sustituyen por nuevas. Las métricas de un determinado espacio de nombres se reemplazan únicamente por métricas del mismo espacio de nombres.

> **Note**
>
> Las métricas no se guardan si el proceso de host para el conector se interrumpe. Por ejemplo, esta interrupción puede ocurrir durante la implementación de grupos o cuando el dispositivo se reinicia.

Nombre para mostrar en la consola AWS IoT: Número máximo de métricas que se deben retener

Obligatorio: `true`

Tipo: `string`

Patrón válido: `^([2-9]\d{3}|[1-9]\d{4,})$`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector de CloudWatch métricas.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyCloudWatchMetricsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/CloudWatchMetrics/
versions/4",
            "Parameters": {
                "PublishInterval" : "600",
                "PublishRegion" : "us-west-2",
                "MemorySize" : "16",
                "MaxMetricsToRetain" : "2500",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de Conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta métricas sobre un tema de MQTT y las publica en. CloudWatch Los mensajes de entrada deben tener un formato JSON válido.

Filtro de temas en la suscripción

`cloudwatch/metric/put`

Propiedades de mensajes

`request`

Información acerca de la métrica en este mensaje.

El objeto de la solicitud contiene los datos de métricas que se publicarán en CloudWatch. Los valores de métricas debe cumplir las especificaciones de la API de `PutMetricData` API. Solo se necesitan las propiedades `namespace`, `metricData.metricName` y `metricData.value`.

Obligatorio: `true`

Escriba: `object` que incluya las siguientes propiedades:

`namespace`

El espacio de nombres definido por el usuario para los datos de las métricas de esta solicitud. CloudWatch utiliza los espacios de nombres como contenedores para los puntos de datos métricos.

> **Note**
>
> No se puede especificar un espacio de nombres que comience la cadena reservada por `AWS/`.

Obligatorio: `true`

Tipo: `string`

Patrón válido: `[^:].*`

`metricData`

Los datos de la métrica.

Obligatorio: `true`

Escriba: `object` que incluya las siguientes propiedades:

`metricName`

El nombre de la métrica.

Obligatorio: `true`

Tipo: `string`

`dimensions`

Las dimensiones que están asociados con la métrica. Las dimensiones proporcionan información adicional acerca de la métrica y sus datos. Una métrica puede definir hasta 10 dimensiones.

Este conector incluye automáticamente una dimensión denominada `coreName`, cuyo valor es el nombre del núcleo.

Obligatorio: `false`

Tipo: `array` de objetos de dimensión que incluyen las siguientes propiedades:

`name`

El nombre de la dimensión.

Obligatorio: `false`

Tipo: `string`

`value`

El valor de la dimensión.

Obligatorio: `false`

Tipo: `string`

`timestamp`

La hora a la que se ha recibido los datos de las métricas, expresado como segundos desde `Jan 1, 1970 00:00:00 UTC`. Si no se especifica este valor, el conector utiliza la hora en que se recibió el mensaje.

Obligatorio: `false`

Tipo: `timestamp`

> **Note**
>
> Si utiliza las versiones 1 y 4 de este conector, le recomendamos que recupere la marca de tiempo por separado para cada métrica cuando envíe varias métricas desde una sola fuente. No utilice una variable para almacenar la marca de tiempo.

`value`

El valor de la métrica.

> **Note**
>
> CloudWatch rechaza los valores que son demasiado pequeños o demasiado grandes. Los valores deben estar en el rango de `8.515920e-109` a `1.174271e+108` (Base 10) o `2e-360` a `2e360` (Base 2). Los valores especiales (por ejemplo, `NaN`, `+Infinity`, `-Infinity`) no son compatibles.

Obligatorio: `true`

Tipo: `double`

`unit`

La unidad de la métrica.

Obligatorio: `false`

Tipo: `string`

Valores válidos: `Seconds, Microseconds, Milliseconds, Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Bits, Kilobits, Megabits, Gigabits, Terabits, Percent, Count, Bytes/Second, Kilobytes/Second, Megabytes/Second, Gigabytes/Second, Terabytes/`

```
        Second, Bits/Second, Kilobits/Second, Megabits/Second, Gigabits/
        Second, Terabits/Second, Count/Second, None
```

Límites

Todos los límites impuestos por la CloudWatch `PutMetricData`API se aplican a las métricas cuando se utiliza este conector. Los siguientes límites son especialmente importantes:

- Límite de 40 KB en la carga de API
- 20 métricas por solicitud de API
- 150 transacciones por segundo (TPS) para la API de `PutMetricData`

Para obtener más información, consulta CloudWatch los límites en la Guía del CloudWatch usuario de Amazon.

Ejemplo de entrada

```
{
   "request": {
       "namespace": "Greengrass",
       "metricData":
           {
               "metricName": "latency",
               "dimensions": [
                   {
                       "name": "hostname",
                       "value": "test_hostname"
                   }
               ],
               "timestamp": 1539027324,
               "value": 123.0,
               "unit": "Seconds"
            }
    }
}
```

Datos de salida

Este conector publica información de estado como datos de salida en un tema MQTT.

Filtro de temas en la suscripción

```
cloudwatch/metric/put/status
```

Ejemplo de salida: Correcto

La respuesta incluye el espacio de nombres de los datos de la métrica y el `RequestId` campo de la CloudWatch respuesta.

```
{
   "response": {
        "cloudwatch_rid":"70573243-d723-11e8-b095-75ff2EXAMPLE",
        "namespace": "Greengrass",
        "status":"success"
    }
}
```

Ejemplo de salida: Error

```
{
   "response" : {
        "namespace": "Greengrass",
        "error": "InvalidInputException",
        "error_message":"cw metric is invalid",
        "status":"fail"
   }
}
```

> **Note**
>
> Si el conector detecta un error que se puede volver a intentar (por ejemplo, errores de conexión), volverá a intentar la publicación en el siguiente lote.

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.

- Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.

   Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de AWS IoT Greengrass Core para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.
3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el conector y configure sus parámetros.

   c. Agregue suscripciones que permitan al conector recibir datos de entrada y enviar datos de salida en filtros de tema compatibles.

      - Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
      - Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.
4. Implemente el grupo.
5. En la consola de AWS IoT, en la página Prueba suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la característica envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'cloudwatch/metric/put'

def create_request_with_all_fields():
    return  {
        "request": {
            "namespace": "Greengrass_CW_Connector",
            "metricData": {
                "metricName": "Count1",
                "dimensions": [
                    {
                        "name": "test",
                        "value": "test"
                    }
                ],
                "value": 1,
                "unit": "Seconds",
                "timestamp": time.time()
            }
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()
```

```
def lambda_handler(event, context):
    return
```

Licencias

El conector CloudWatch Metrics incluye el siguiente software o licencia de terceros:

- AWS SDK for Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registros de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
5	Se corrigió para añadir la admisión de marcas de tiempo duplicadas en los datos de entrada.
4	Se ha agregado el parámetro `Isolation Mode` para configurar el modo de creación de contenedores del conector.
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se ha introducido una corrección para reducir el registro excesivo.

Versión	Cambios
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- Uso de CloudWatch las métricas de Amazon en la Guía del CloudWatch usuario de Amazon
- PutMetricDataen la referencia de la CloudWatch API de Amazon

Conector de Device Defender

El conector Device Defender notifica a los administradores de los cambios en el estado de un dispositivo del núcleo de Greengrass. Esto puede ayudar a identificar comportamiento inusual que podrían indicar un dispositivo en riesgo.

Este conector lee las métricas del sistema desde el directorio `/proc` en el dispositivo de núcleo y, a continuación, publica las métricas en AWS IoT Device Defender. Para obtener detalles sobre los informes de métricas, consulte Especificación de documentos de métricas de dispositivos en la Guía del desarrollador de AWS IoT.

Este conector tiene las siguientes versiones.

Versión	ARN
3	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/3`

Versión	ARN
2.	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/2`
1	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

  > **Note**
  >
  > Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- AWS IoT Device Defender configurado para utilizar la función de detección para realizar un seguimiento de infracciones. Para obtener más información, consulte Detectar en la Guía para desarrolladores de AWS IoT.
- Un recurso de volumen local en el grupo de Greengrass que apunta al directorio `/proc`. El recurso debe usar las siguientes propiedades:

  - Ruta de origen: `/proc`
  - Ruta de destino: `/host_proc` (o un valor que coincide con el patrón válido)
  - AutoAddGroupOwner: `true`
- La biblioteca psutil instalada en el núcleo de Greengrass. La versión 5.7.0 es la última versión que se verifica para que funcione con el conector.
- La biblioteca cbor instalada en el núcleo de Greengrass. La versión 1.0.0 es la última versión que se verifica para que funcione con el conector.

Versions 1 - 2

- Software AWS IoT Greengrass Core versión 1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- AWS IoT Device Defender configurado para utilizar la función de detección para realizar un seguimiento de infracciones. Para obtener más información, consulte Detectar en la Guía para desarrolladores de AWS IoT.
- Un recurso de volumen local en el grupo de Greengrass que apunta al directorio `/proc`. El recurso debe usar las siguientes propiedades:
  - Ruta de origen: `/proc`
  - Ruta de destino: `/host_proc` (o un valor que coincide con el patrón válido)
  - AutoAddGroupOwner: `true`
- La biblioteca psutil instalada en el núcleo de Greengrass.
- La biblioteca cbor instalada en el núcleo de Greengrass.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

`SampleIntervalSeconds`

El número de segundos entre cada ciclo de recopilación y generación de informes de métricas. El valor mínimo es de 300 segundos (5 minutos).

Nombre para mostrar en la consola AWS IoT: Intervalo de informes de métricas

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]*(?:3[0-9][0-9]|[4-9][0-9]{2}|[1-9][0-9]{3,})$`

`ProcDestinationPath-ResourceId`

El ID del recurso del volumen `/proc`.

> **Note**
>
> A este conector se le concede acceso de solo lectura al recurso.

Nombre para mostrar en la consola AWS IoT: Recurso para el directorio /proc

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[a-zA-Z0-9_-]+`

`ProcDestinationPath`

La ruta de destino del recurso del volumen `/proc`.

Nombre para mostrar en la consola AWS IoT: Ruta de destino del recurso /proc

Obligatorio: `true`

Escriba: `string`

Patrón válido: `\/[a-zA-Z0-9_-]+`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector de Device Defender.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
```

```
    "Connectors": [
        {
            "Id": "MyDeviceDefenderConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/DeviceDefender/
versions/3",
            "Parameters": {
                "SampleIntervalSeconds": "600",
                "ProcDestinationPath": "/host_proc",
                "ProcDestinationPath-ResourceId": "my-proc-resource"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de Conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector no acepta mensajes MQTT como datos de entrada.

Datos de salida

Este conector publica métricas de seguridad en AWS IoT Device Defender como datos de salida.

Filtro de temas en la suscripción

```
$aws/things/+/defender/metrics/json
```

> **Note**
>
> Esta es la sintaxis del tema que AWS IoT Device Defender espera. El conector sustituye el comodín + con el nombre de dispositivo (por ejemplo `$aws/things/thing-name/defender/metrics/json`).

Ejemplo de salida

Para obtener detalles sobre los informes de métricas, consulte Especificación de documentos de métricas de dispositivos en la Guía del desarrolador de AWS IoT.


```
{
    "header": {
        "report_id": 1529963534,
        "version": "1.0"
    },
    "metrics": {
        "listening_tcp_ports": {
            "ports": [
                {
                    "interface": "eth0",
                    "port": 24800
                },
                {
                    "interface": "eth0",
                    "port": 22
                },
                {
                    "interface": "eth0",
                    "port": 53
                }
            ],
            "total": 3
        },
        "listening_udp_ports": {
            "ports": [
                {
                    "interface": "eth0",
                    "port": 5353
                },
                {
                    "interface": "eth0",
                    "port": 67
                }
            ],
            "total": 2
        },
        "network_stats": {
            "bytes_in": 1157864729406,
            "bytes_out": 1170821865,
```

```
            "packets_in": 693092175031,
            "packets_out": 738917180
        },
        "tcp_connections": {
            "established_connections":{
                "connections": [
                    {
                    "local_interface": "eth0",
                    "local_port": 80,
                    "remote_addr": "192.168.0.1:8000"
                    },
                    {
                    "local_interface": "eth0",
                    "local_port": 80,
                    "remote_addr": "192.168.0.1:8000"
                    }
                ],
                "total": 2
            }
        }
    }
}
```

Licencias

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se ha introducido una corrección para reducir el registro excesivo.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- Device Defender en la Guía para desarrolladores de AWS IoT

Conector de implementación de aplicaciones Docker

El conector de implementación de aplicaciones de Docker de Greengrass facilita la ejecución de las imágenes de Docker en un AWS IoT Greengrass core. El conector utiliza Docker Compose para iniciar una aplicación Docker multicontenedor desde un archivo de `docker-compose.yml`. Concretamente, el conector ejecuta comandos de `docker-compose` para administrar los contenedores Docker en un único dispositivo central. Para obtener más información, consulte Overview of Greengrass group role permissions (Descripción general de Docker Compose en la documentación de Docker. El conector puede acceder a imágenes de Docker almacenadas en registros de contenedores Docker, como Amazon Elastic Container Registry (Amazon ECR), Docker Hub y registros privados de confianza de Docker.

Después de implementar el grupo de Greengrass, el conector lanza las últimas imágenes e inicia los contenedores de Docker. Ejecuta el `docker-compose pull` comando y `docker-compose up`. El conctor publica el comando de y, a continuación, publica un tema MQTT de salida. También registra información del estado sobre contenedores Docker en funcionamiento. Esto te permite supervisar los registros de tus aplicaciones en Amazon CloudWatch. Para obtener más información, consulte the section called “Monitorización con registros de AWS IoT Greengrass”. El conector también inicia contenedores Docker cada vez que se reinicia el daemon de Greengrass. La cantidad de contenedores Docker que se puede ejecutar en el núcleo depende del hardware que tenga.

Los contenedores Docker se ejecutan fuera del dominio de Greengrass en el dispositivo central, de modo que no pueden acceder a la comunicación entre procesos (IPC) del núcleo. Sin embargo, puede configurar algunos canales de comunicación con componentes de Greengrass, como las funciones locales de Lambda. Para obtener más información, consulte the section called “Comunicación con contenedores Docker”.

Puede usar el conector para alojar un servidor web o un servidor MySQL en su dispositivo central. Los servicios locales de sus aplicaciones Docker pueden comunicarse entre sí, con otros procesos del entorno local y con servicios en la nube. Por ejemplo, puede ejecutar un servidor web del núcleo que envíe solicitudes desde las funciones de Lambda a un servicio web en la nube.

Este conector se ejecuta en modo de aislamiento Sin contenedor, por lo que puede implementarlo en un grupo de Greengrass que se ejecute sin creación de contenedores de Greengrass.

Este conector tiene las siguientes versiones.

Versión	ARN
7	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/7`
6	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/6`
5	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/5`
4	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/4`
3	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/3`
2.	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/2`
1	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

- Software AWS IoT Greengrass Core versión 1.10 o posterior.

  > **Note**
  >
  > Este conector no es compatible con las OpenWrt distribuciones.

- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

  > **Note**
  >
  > Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Un mínimo de 36 MB de RAM en el núcleo de Greengrass para que el conector supervise los contenedores Docker en funcionamiento. El requisito total de memoria depende del número de contenedores Docker que se ejecuten en el núcleo.
- Docker Engine versión 1.9.1 o posterior instalado en el núcleo de Greengrass. La versión 19.0.3 es la última versión cuyo funcionamiento con el conector se ha verificado.

  El ejecutable `docker` debe estar en el directorio `/usr/bin` o `/usr/local/bin`.

> **Important**
>
> Le recomendamos que instale un almacén de credenciales para proteger las copias locales de sus credenciales de Docker. Para obtener más información, consulte the section called “Notas de seguridad”.

Para obtener información sobre la instalación de Docker en distribuciones de Amazon Linux, consulte Conceptos básicos de Docker para Amazon ECS en la Guía para desarrolladores de Amazon Elastic Container Service.

- Docker Compose instalado en el núcleo de Greengrass. El ejecutable `docker-compose` debe estar en el directorio `/usr/bin` o `/usr/local/bin`.

  Las siguientes versiones de Docker Compose se verifican para que funcionen con el conector.

Versión del conector	Versión de Docker Compose verificada
7	1.25.4
6	1.25.4
5	1.25.4
4	1.25.4
3	1.25.4
2	1.25.1
1	1.24.1

- Un único archivo de Docker Compose (por ejemplo, `docker-compose.yml`), almacenado en Amazon Simple Storage Service (Amazon S3). El formato debe ser compatible con la versión de Docker Compose instalada en el núcleo. Debe probar el archivo antes de usarlo en su núcleo. Si edita el archivo después de implementar el grupo de Greengrass, debe volver a implementar el grupo para actualizar la copia local del núcleo.

- Un usuario de Linux con permiso para llamar al daemon local de Docker y escribir en el directorio que almacena la copia local del archivo de Compose. Para obtener más información, consulte Configuración del usuario de Docker en el núcleo.
- El rol de grupo Greengrass configurado para permitir la acción `s3:GetObject` en el bucket S3 que contiene el archivo de Compose. Este permiso se muestra en la siguiente política de ejemplo de política de IAM.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToComposeFileS3Bucket",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::bucket-name/*"
        }
    ]
}
```

> **Note**
>
> Si su bucket de S3 está habilitado para el control de versiones, entonces el rol debe estar configurado para permitir también la acción `s3:GetObjectVersion`. Para obtener más información, consulte Uso del control de versiones en la Guía del usuario de Amazon Simple Storage Service.

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

- Si el archivo Docker Compose hace referencia a una imagen Docker almacenada en Amazon ECR, el rol de grupo Greengrass configurado para permitir lo siguiente:
  - Las acciones de `ecr:GetDownloadUrlForLayer` y `ecr:BatchGetImage` en los repositorios de Amazon ECR que contienen las imágenes de Docker.

- La acción de `ecr:GetAuthorizationToken` en sus recursos.

Los repositorios deben estar en la misma Cuenta de AWS y Región de AWS que el conector.

> **Important**
>
> Todos los conectores y las funciones de Lambda del grupo de Greengrass pueden asumir permisos en el rol de grupo. Para obtener más información, consulte the section called "Notas de seguridad".

Estos permisos se muestran en la siguiente política de ejemplo.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGetEcrRepositories",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:region:account-id:repository/repository-name"
            ]
        },
        {
            "Sid": "AllowGetEcrAuthToken",
            "Effect": "Allow",
            "Action": "ecr:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```

Para obtener más información, consulte Ejemplos de políticas de repositorio de Amazon ECR en la Guía del usuario de Amazon ECR.

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the

section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

- Si el archivo de Docker Compose hace referencia a una imagen de Docker de AWS Marketplace, el conector también tiene los siguientes requisitos:
  - Debe estar suscrito a los productos de contenedor de AWS Marketplace. Para obtener más información, consulte Búsqueda y suscripción a productos de contenedor en la Guía del suscriptor de AWS Marketplace.
  - AWS IoT Greengrass debe configurarse para admitir secretos locales, como se describe en Requisitos de secretos. El conector utiliza esta función únicamente para recuperar sus secretos de AWS Secrets Manager, no para almacenarlos.
  - Debe crear un secreto en Secrets Manager para cada registro de AWS Marketplace que almacene una imagen de Docker a la que se hace referencia en su archivo de Compose. Para obtener más información, consulte the section called “Acceso a imágenes de Docker desde repositorios privados”.
- Si el archivo de Docker Compose hace referencia a una imagen de Docker de repositorios privados en registros distintos de Amazon ECR, como Docker Hub, el conector también tendrá los siguientes requisitos:
  - AWS IoT Greengrass debe configurarse para admitir secretos locales, como se describe en Requisitos de secretos. El conector utiliza esta función únicamente para recuperar sus secretos de AWS Secrets Manager, no para almacenarlos.
  - Debe crear un secreto en Secrets Manager para cada repositorio privado que almacene una imagen de Docker a la que se hace referencia en su archivo de Compose. Para obtener más información, consulte the section called “Acceso a imágenes de Docker desde repositorios privados”.
- El daemon de Docker debe ejecutarse cuando implemente un grupo de Greengrass que contenga este conector.

Acceso a imágenes de Docker desde repositorios privados

Si utiliza credenciales para acceder a sus imágenes de Docker, debe permitir que el conector tenga acceso a ellas. La forma en que lo haga depende de dónde se encuentre la imagen de Docker.

En el caso de las imágenes de Docker almacenadas en Amazon ECR, concede permiso para obtener su token de autorización en el rol del grupo de Greengrass. Para obtener más información, consulte the section called “Requisitos”.

En el caso de las imágenes de Docker almacenadas en otros repositorios o registros privados, debe crear un secreto en AWS Secrets Manager para almacenar su información de inicio de sesión. Lo que incluye las imágenes de Docker a las que se ha suscrito en AWS Marketplace. Cree un secreto para cada repositorio. Si actualiza sus secretos en Secrets Manager, los cambios se propagarán al núcleo la próxima vez que implemente el grupo.

> **Note**
>
> Secrets Manager es un servicio que puede utilizar para almacenar y administrar de forma segura sus credenciales, claves y otros secretos en la Nube de AWS. Para obtener más información, consulte ¿Qué es AWS Secrets Manager? en la Guía del usuario de AWS Secrets Manager.

Cada secreto debe contener las siguientes claves:

Clave	Valor
`username`	El nombre de usuario utilizado para acceder al repositorio o al registro.
`password`	La contraseña utilizada para acceder al repositorio o al registro.
`registryUrl`	El punto de enlace del registro. Este debe coincidir con la URL de registro correspon diente del archivo de Compose.

> **Note**
>
> Para permitir el acceso de AWS IoT Greengrass a un secreto de forma predeterminada, el nombre del secreto debe comenzar por greengrass-. De lo contrario, debe conceder el acceso su rol de servicio de Greengrass. Para obtener más información, consulte the section called “Permitir que AWS IoT Greengrass obtenga valores de secretos”.

Para obtener información de inicio de sesión para las imágenes de Docker desde AWS Marketplace

1. Obtenga su contraseña para las imágenes de Docker de AWS Marketplace mediante el comando `aws ecr get-login-password`. Para obtener más información, consulte get-login-password en la Referencia de los comandos de AWS CLI.

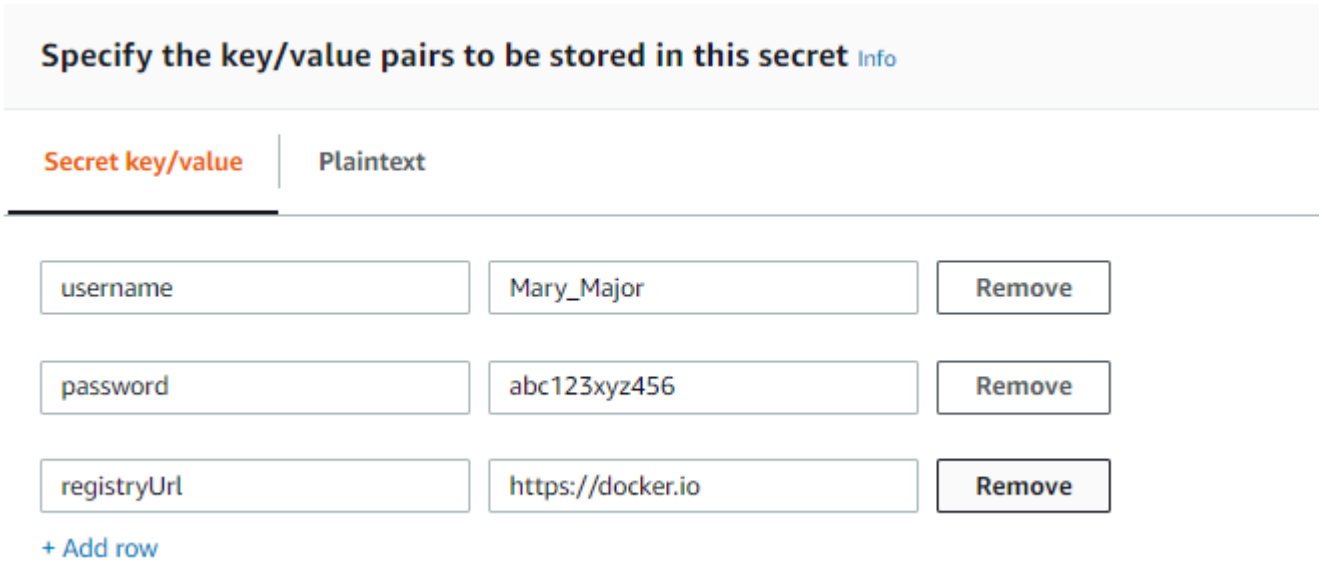
   ```
   aws ecr get-login-password
   ```

2. Recupere la URL de registro de la imagen de Docker. Abra el sitio web AWS Marketplace y abra la página de lanzamiento del producto contenedor. En Imágenes del contenedor, seleccione Ver detalles de la imagen del contenedor para buscar el nombre de usuario y la URL del registro.

Use el nombre de usuario, la contraseña y la URL del registro recuperados para crear un secreto para cada registro AWS Marketplace que almacene las imágenes de Docker a las que se hace referencia en tu archivo de Compose.

Para crear secretos (consola)

En la consola de AWS Secrets Manager, seleccione Other type of secrets (Otro tipo de secretos). En Specify the key-value pairs to be stored for this secret (Especificar los pares clave-valor que se van a almacenar para este secreto), añada filas para `username`, `password` y `registryUrl`. Para obtener más información, consulte Creación de un secreto básico en la Guía del usuarioAWS Secrets Manager.



Para crear secretos (CLI)

Utilice AWS CLIel comando Secrets Manager de CloudFront en la `create-secret` como se muestra en el siguiente ejemplo. Para obtener más información, consulte create-cluster en la Referencia de comandos de la AWS CLI.

```
aws secretsmanager create-secret --name greengrass-MySecret --secret-string
 [{"username":"Mary_Major"},{"password":"abc123xyz456"},{"registryUrl":"https://
docker.io"}]
```

Important

Es su responsabilidad proteger el directorio `DockerComposeFileDestinationPath` en el que se guarda el archivo de Docker Compose y las credenciales de las imágenes de Docker de los repositorios privados. Para obtener más información, consulte the section called “Notas de seguridad”.

Parámetros

Este conector proporciona los siguientes parámetros:

Version 7

`DockerComposeFileS3Bucket`

El nombre del bucket S3 que contiene el archivo de Docker Compose. Cuando cree el bucket, asegúrese de seguir las reglas para nombres de bucket descritos en la guía del desarrollador de Amazon Simple Storage Service.

Nombre que mostrar en la consola AWS IoT: Archivo de Docker Compose en S3

Note

En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clave de objeto de su archivo de Docker Compose en Amazon S3. Para obtener más información, incluidos los límites y directrices de nomenclatura, consulte Clave de objetos y metadatos de objeto en la guía del desarrollador de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileS3Version`

La versión de objeto del archivo de Docker Compose en Amazon S3. Para obtener más información, incluidas las pautas de denominación de las claves de objeto, consulte Uso del control de versiones en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `false`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileDestinationPath`

La ruta absoluta del directorio local utilizado para almacenar una copia del archivo de Docker Compose. Debe ser un directorio existente. El usuario especificado para `DockerUserId` debe tener permiso para crear un archivo en este directorio. Para obtener más información, consulte the section called “Configuración del usuario de Docker en el núcleo”.

> **Important**
>
> Este directorio almacena las credenciales y el archivo de Docker Compose de sus imágenes de Docker de repositorios privados. Es su responsabilidad proteger este directorio. Para obtener más información, consulte the section called “Notas de seguridad”.

Nombre que mostrar en la consola AWS IoT: Ruta del directorio del archivo Compose local

Obligatorio: `true`

Tipo: `string`

Patrón válido `\/.*\/?`

Ejemplo: `/home/username/myCompose`

`DockerUserId`

El ID del usuario de Linux con el que se ejecuta el conector. Este usuario debe pertenecer al grupo de Linux de `docker` en el dispositivo central y tener permisos de escritura en el directorio de `DockerComposeFileDestinationPath`. Para obtener más información, consulte Configuración del usuario de Docker en el núcleo.

> **Note**
>
> Se recomienda evitar la ejecución como raíz a menos que sea absolutamente necesario. Si especifica el usuario raíz, debe permitir que las funciones de Lambda se ejecuten como raíz en AWS IoT Greengrass. Para obtener más información, consulte the section called “Ejecución de una función de Lambda como raíz”.

Nombre que mostrar en la consola AWS IoT: ID de usuario de Docker

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Los nombres de recursos de Amazon (ARN) de los secretos de AWS Secrets Manager que contienen la información de inicio de sesión utilizada para acceder a las imágenes de Docker en repositorios privados. Para obtener más información, consulte the section called “Acceso a imágenes de Docker desde repositorios privados”.

Nombre que mostrar en la consola AWS IoT: Credenciales para repositorios privados

Obligatorio: `false`. Este parámetro es necesario para acceder a las imágenes de Docker almacenadas en repositorios privados.

Tipo: `array` de `string`

Patrón válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

La frecuencia (en segundos) a la que el conector registra información de estado sobre los contenedores Docker que se ejecutan en el núcleo. El valor predeterminado es 300 segundos (5 minutos).

Nombre que mostrar en la consola AWS IoT: Frecuencia de registro

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica si se aplicará forzosamente la implementación de Docker en caso de que se produzca un error por una limpieza incorrecta de la última implementación. El valor predeterminado es `False`.

Nombre que mostrar en la consola AWS IoT: Forzar implementación

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

`DockerPullBeforeUp`

Indica si el implementador debe ejecutarse `docker-compose pull` antes de ejecutarse `docker-compose up` para detectar un pull-down-up comportamiento. El valor predeterminado es `True`.

Nombre que mostrar en la consola AWS IoT: Extraer antes de levantar el Docker

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

`StopContainersOnNewDeployment`

Indica si el conector debe detener los contenedores docker gestionados por Docker Deployer cuando se detiene el GGC (el GGC se detiene cuando se despliega un grupo nuevo o se cierra el núcleo). El valor predeterminado es `True`.

Nombre que mostrar en la consola AWS IoT: Docker se detiene en una nueva implementación

> **Note**
>
> Recomendamos mantener este parámetro establecido en su valor `True` predeterminado. El parámetro para `False` hace que el contenedor de Docker siga ejecutándose incluso después de terminar el AWS IoT Greengrass core o iniciar una nueva implementación. Si establece este parámetro en `False`, debe asegurarse de que sus contenedores de Docker se mantengan según sea necesario en caso de que se añada o cambie el nombre del servicio `docker-compose`.
> Para obtener más información, consulte Componentes de en la documentación de `docker-compose`.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

`DockerOfflineMode`

Indica si se debe utilizar el archivo Docker Compose existente al iniciar AWS IoT Greengrass sin conexión. El valor predeterminado es `False`.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

Version 6

`DockerComposeFileS3Bucket`

El nombre del bucket S3 que contiene el archivo de Docker Compose. Cuando cree el bucket, asegúrese de seguir las reglas para nombres de bucket descritos en la guía del desarrollador de Amazon Simple Storage Service.

Nombre que mostrar en la consola AWS IoT: Archivo de Docker Compose en S3

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clave de objeto de su archivo de Docker Compose en Amazon S3. Para obtener más información, incluidos los límites y directrices de nomenclatura, consulte Clave de objetos y metadatos de objeto en la guía del desarrollador de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileS3Version`

La versión de objeto del archivo de Docker Compose en Amazon S3. Para obtener más información, incluidas las pautas de denominación de las claves de objeto, consulte Uso del control de versiones en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `false`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileDestinationPath`

La ruta absoluta del directorio local utilizado para almacenar una copia del archivo de Docker Compose. Debe ser un directorio existente. El usuario especificado para `DockerUserId` debe tener permiso para crear un archivo en este directorio. Para obtener más información, consulte the section called “Configuración del usuario de Docker en el núcleo”.

> **Important**
>
> Este directorio almacena las credenciales y el archivo de Docker Compose de sus imágenes de Docker de repositorios privados. Es su responsabilidad proteger este directorio. Para obtener más información, consulte the section called “Notas de seguridad”.

Nombre que mostrar en la consola AWS IoT: Ruta del directorio del archivo Compose local

Obligatorio: `true`

Tipo: `string`

Patrón válido `\/.*\/?`

Ejemplo: `/home/username/myCompose`

`DockerUserId`

El ID del usuario de Linux con el que se ejecuta el conector. Este usuario debe pertenecer al grupo de Linux de `docker` en el dispositivo central y tener permisos de escritura en el directorio de `DockerComposeFileDestinationPath`. Para obtener más información, consulte Configuración del usuario de Docker en el núcleo.

> **Note**
>
> Se recomienda evitar la ejecución como raíz a menos que sea absolutamente necesario. Si especifica el usuario raíz, debe permitir que las funciones de Lambda se ejecuten como raíz en AWS IoT Greengrass. Para obtener más información, consulte the section called “Ejecución de una función de Lambda como raíz”.

Nombre que mostrar en la consola AWS IoT: ID de usuario de Docker

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Los nombres de recursos de Amazon (ARN) de los secretos de AWS Secrets Manager que contienen la información de inicio de sesión utilizada para acceder a las imágenes de Docker en repositorios privados. Para obtener más información, consulte the section called “Acceso a imágenes de Docker desde repositorios privados”.

Nombre que mostrar en la consola AWS IoT: Credenciales para repositorios privados

Obligatorio: `false`. Este parámetro es necesario para acceder a las imágenes de Docker almacenadas en repositorios privados.

Tipo: `array` de `string`

Patrón válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

La frecuencia (en segundos) a la que el conector registra información de estado sobre los contenedores Docker que se ejecutan en el núcleo. El valor predeterminado es 300 segundos (5 minutos).

Nombre que mostrar en la consola AWS IoT: Frecuencia de registro

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica si se aplicará forzosamente la implementación de Docker en caso de que se produzca un error por una limpieza incorrecta de la última implementación. El valor predeterminado es `False`.

Nombre que mostrar en la consola AWS IoT: Forzar implementación

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

`DockerPullBeforeUp`

Indica si el implementador debe ejecutarse `docker-compose pull` antes de ejecutarse `docker-compose up` para un pull-down-up comportamiento. El valor predeterminado es `True`.

Nombre que mostrar en la consola AWS IoT: Extraer antes de levantar el Docker

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

`StopContainersOnNewDeployment`

Indica si el conector debe detener los contenedores docker gestionados por Docker Deployer cuando se detiene el GGC (cuando se realiza una implementación de un nuevo grupo o se apaga el núcleo). El valor predeterminado es `True`.

Nombre que mostrar en la consola AWS IoT: Docker se detiene en una nueva implementación

> **Note**
>
> Recomendamos mantener este parámetro establecido en su valor `True` predeterminado. El parámetro para `False` hace que el contenedor de Docker siga ejecutándose incluso después de terminar el AWS IoT Greengrass core o iniciar una nueva implementación. Si establece este parámetro en `False`, debe asegurarse de que sus contenedores de Docker se mantengan según sea necesario en caso de que se añada o cambie el nombre del servicio `docker-compose`.
> Para obtener más información, consulte Componentes de en la documentación de `docker-compose`.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

Version 5

`DockerComposeFileS3Bucket`

El nombre del bucket S3 que contiene el archivo de Docker Compose. Cuando cree el bucket, asegúrese de seguir las reglas para nombres de bucket descritos en la guía del desarrollador de Amazon Simple Storage Service.

Nombre que mostrar en la consola AWS IoT: Archivo de Docker Compose en S3

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clave de objeto de su archivo de Docker Compose en Amazon S3. Para obtener más información, incluidos los límites y directrices de nomenclatura, consulte Clave de objetos y metadatos de objeto en la guía del desarrollador de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileS3Version`

La versión de objeto del archivo de Docker Compose en Amazon S3. Para obtener más información, incluidas las pautas de denominación de las claves de objeto, consulte Uso del control de versiones en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `false`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileDestinationPath`

La ruta absoluta del directorio local utilizado para almacenar una copia del archivo de Docker Compose. Debe ser un directorio existente. El usuario especificado para `DockerUserId` debe tener permiso para crear un archivo en este directorio. Para obtener más información, consulte the section called “Configuración del usuario de Docker en el núcleo”.

> **Important**
>
> Este directorio almacena las credenciales y el archivo de Docker Compose de sus imágenes de Docker de repositorios privados. Es su responsabilidad proteger este directorio. Para obtener más información, consulte the section called “Notas de seguridad”.

Nombre que mostrar en la consola AWS IoT: Ruta del directorio del archivo Compose local

Obligatorio: `true`

Tipo: `string`

Patrón válido `\/.*\/?`

Ejemplo: `/home/username/myCompose`

`DockerUserId`

El ID del usuario de Linux con el que se ejecuta el conector. Este usuario debe pertenecer al grupo de Linux de `docker` en el dispositivo central y tener permisos de escritura en el directorio de `DockerComposeFileDestinationPath`. Para obtener más información, consulte Configuración del usuario de Docker en el núcleo.

> **Note**
>
> Se recomienda evitar la ejecución como raíz a menos que sea absolutamente necesario. Si especifica el usuario raíz, debe permitir que las funciones de Lambda se ejecuten como raíz en AWS IoT Greengrass. Para obtener más información, consulte the section called “Ejecución de una función de Lambda como raíz”.

Nombre que mostrar en la consola AWS IoT: ID de usuario de Docker

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Los nombres de recursos de Amazon (ARN) de los secretos de AWS Secrets Manager que contienen la información de inicio de sesión utilizada para acceder a las imágenes de Docker en repositorios privados. Para obtener más información, consulte the section called “Acceso a imágenes de Docker desde repositorios privados”.

Nombre que mostrar en la consola AWS IoT: Credenciales para repositorios privados

Obligatorio: `false`. Este parámetro es necesario para acceder a las imágenes de Docker almacenadas en repositorios privados.

Tipo: `array` de `string`

Patrón válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

La frecuencia (en segundos) a la que el conector registra información de estado sobre los contenedores Docker que se ejecutan en el núcleo. El valor predeterminado es 300 segundos (5 minutos).

Nombre que mostrar en la consola AWS IoT: Frecuencia de registro

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica si se aplicará forzosamente la implementación de Docker en caso de que se produzca un error por una limpieza incorrecta de la última implementación. El valor predeterminado es `False`.

Nombre que mostrar en la consola AWS IoT: Forzar implementación

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

`DockerPullBeforeUp`

Indica si el implementador debe ejecutarse `docker-compose pull` antes de ejecutarse `docker-compose up` para un pull-down-up comportamiento. El valor predeterminado es `True`.

Nombre que mostrar en la consola AWS IoT: Extraer antes de levantar el Docker

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

Versions 2 - 4

`DockerComposeFileS3Bucket`

El nombre del bucket S3 que contiene el archivo de Docker Compose. Cuando cree el bucket, asegúrese de seguir las reglas para nombres de bucket descritos en la guía del desarrollador de Amazon Simple Storage Service.

Nombre que mostrar en la consola AWS IoT: Archivo de Docker Compose en S3

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clave de objeto de su archivo de Docker Compose en Amazon S3. Para obtener más información, incluidos los límites y directrices de nomenclatura, consulte Clave de objetos y metadatos de objeto en la guía del desarrollador de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileS3Version`

La versión de objeto del archivo de Docker Compose en Amazon S3. Para obtener más información, incluidas las pautas de denominación de las claves de objeto, consulte Uso del control de versiones en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `false`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileDestinationPath`

La ruta absoluta del directorio local utilizado para almacenar una copia del archivo de Docker Compose. Debe ser un directorio existente. El usuario especificado para `DockerUserId` debe tener permiso para crear un archivo en este directorio. Para obtener más información, consulte the section called “Configuración del usuario de Docker en el núcleo”.

> **Important**
>
> Este directorio almacena las credenciales y el archivo de Docker Compose de sus imágenes de Docker de repositorios privados. Es su responsabilidad proteger este directorio. Para obtener más información, consulte the section called “Notas de seguridad”.

Nombre que mostrar en la consola AWS IoT: Ruta del directorio del archivo Compose local

Obligatorio: `true`

Tipo: `string`

Patrón válido `\/.*\/?`

Ejemplo: `/home/username/myCompose`

`DockerUserId`

El ID del usuario de Linux con el que se ejecuta el conector. Este usuario debe pertenecer al grupo de Linux de `docker` en el dispositivo central y tener permisos de escritura en el directorio de `DockerComposeFileDestinationPath`. Para obtener más información, consulte Configuración del usuario de Docker en el núcleo.

> **Note**
>
> Se recomienda evitar la ejecución como raíz a menos que sea absolutamente necesario. Si especifica el usuario raíz, debe permitir que las funciones de Lambda se ejecuten como raíz en AWS IoT Greengrass. Para obtener más información, consulte the section called “Ejecución de una función de Lambda como raíz”.

Nombre que mostrar en la consola AWS IoT: ID de usuario de Docker

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Los nombres de recursos de Amazon (ARN) de los secretos de AWS Secrets Manager que contienen la información de inicio de sesión utilizada para acceder a las imágenes de Docker en repositorios privados. Para obtener más información, consulte the section called “Acceso a imágenes de Docker desde repositorios privados”.

Nombre que mostrar en la consola AWS IoT: Credenciales para repositorios privados

Obligatorio: `false`. Este parámetro es necesario para acceder a las imágenes de Docker almacenadas en repositorios privados.

Tipo: `array` de `string`

Patrón válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

La frecuencia (en segundos) a la que el conector registra información de estado sobre los contenedores Docker que se ejecutan en el núcleo. El valor predeterminado es 300 segundos (5 minutos).

Nombre que mostrar en la consola AWS IoT: Frecuencia de registro

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica si se aplicará forzosamente la implementación de Docker en caso de que se produzca un error por una limpieza incorrecta de la última implementación. El valor predeterminado es `False`.

Nombre que mostrar en la consola AWS IoT: Forzar implementación

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^(true|false)$`

Version 1

`DockerComposeFileS3Bucket`

El nombre del bucket S3 que contiene el archivo de Docker Compose. Cuando cree el bucket, asegúrese de seguir las reglas para nombres de bucket descritos en la guía del desarrollador de Amazon Simple Storage Service.

Nombre que mostrar en la consola AWS IoT: Archivo de Docker Compose en S3

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clave de objeto de su archivo de Docker Compose en Amazon S3. Para obtener más información, incluidos los límites y directrices de nomenclatura, consulte Clave de objetos y metadatos de objeto en la guía del desarrollador de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `true`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileS3Version`

La versión de objeto del archivo de Docker Compose en Amazon S3. Para obtener más información, incluidas las pautas de denominación de las claves de objeto, consulte Uso del control de versiones en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> En la consola, el archivo de Docker Compose en la propiedad S3 combina los parámetros de `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key`, y `DockerComposeFileS3Version`.

Obligatorio: `false`

Tipo: `string`

Patrón válido `.+`

`DockerComposeFileDestinationPath`

La ruta absoluta del directorio local utilizado para almacenar una copia del archivo de Docker Compose. Debe ser un directorio existente. El usuario especificado para `DockerUserId` debe tener permiso para crear un archivo en este directorio. Para obtener más información, consulte the section called “Configuración del usuario de Docker en el núcleo”.

> **Important**
>
> Este directorio almacena las credenciales y el archivo de Docker Compose de sus imágenes de Docker de repositorios privados. Es su responsabilidad proteger este directorio. Para obtener más información, consulte the section called “Notas de seguridad”.

Nombre que mostrar en la consola AWS IoT: Ruta del directorio del archivo Compose local

Obligatorio: `true`

Tipo: `string`

Patrón válido `\/.*\/?`

Ejemplo: `/home/username/myCompose`

`DockerUserId`

El ID del usuario de Linux con el que se ejecuta el conector. Este usuario debe pertenecer al grupo de Linux de `docker` en el dispositivo central y tener permisos de escritura en el

directorio de `DockerComposeFileDestinationPath`. Para obtener más información, consulte Configuración del usuario de Docker en el núcleo.

> **Note**
>
> Se recomienda evitar la ejecución como raíz a menos que sea absolutamente necesario. Si especifica el usuario raíz, debe permitir que las funciones de Lambda se ejecuten como raíz en AWS IoT Greengrass. Para obtener más información, consulte the section called “Ejecución de una función de Lambda como raíz”.

Nombre que mostrar en la consola AWS IoT: ID de usuario de Docker

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Los nombres de recursos de Amazon (ARN) de los secretos de AWS Secrets Manager que contienen la información de inicio de sesión utilizada para acceder a las imágenes de Docker en repositorios privados. Para obtener más información, consulte the section called “Acceso a imágenes de Docker desde repositorios privados”.

Nombre que mostrar en la consola AWS IoT: Credenciales para repositorios privados

Obligatorio: `false`. Este parámetro es necesario para acceder a las imágenes de Docker almacenadas en repositorios privados.

Tipo: `array` de `string`

Patrón válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

La frecuencia (en segundos) a la que el conector registra información de estado sobre los contenedores Docker que se ejecutan en el núcleo. El valor predeterminado es 300 segundos (5 minutos).

Nombre que mostrar en la consola AWS IoT: Frecuencia de registro

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^[1-9]{1}[0-9]{0,3}$`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector de implementación de Greengrass Docker.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyDockerAppplicationDeploymentConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
DockerApplicationDeployment/versions/5",
            "Parameters": {
                "DockerComposeFileS3Bucket": "myS3Bucket",
                "DockerComposeFileS3Key": "production-docker-compose.yml",
                "DockerComposeFileS3Version": "123",
                "DockerComposeFileDestinationPath": "/home/username/myCompose",
                "DockerUserId": "1000",
                "AWSSecretsArnList": "[\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret1-hash\",\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret2-hash\"]",
                "DockerContainerStatusLogFrequency": "30",
                "ForceDeploy": "True",
                "DockerPullBeforeUp": "True"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

Datos de entrada

Este conector no requiere ni acepta datos de entrada.

Datos de salida

Este conector publica el estado del comando de `docker-compose up` como datos de salida.

Filtro de temas en la suscripción

`dockerapplicationdeploymentconnector/message/status`

Ejemplo de salida: Correcto

```
{
  "status":"success",
  "GreengrassDockerApplicationDeploymentStatus":"Successfully triggered docker-
compose up",
  "S3Bucket":"myS3Bucket",
  "ComposeFileName":"production-docker-compose.yml",
  "ComposeFileVersion":"123"
}
```

Ejemplo de salida: Error

```
{
  "status":"fail",
  "error_message":"description of error",
  "error":"InvalidParameter"
}
```

El tipo de error puede ser `InvalidParameter` o `InternalError`.

Configuración del usuario de Docker en el núcleo de AWS IoT Greengrass

El conector de implementación de aplicaciones Docker de Greengrass se ejecuta como el usuario que especifique para el parámetro `DockerUserId`. Si no especifica un valor, el conector se ejecutará como `ggc_user`, que es la identidad de acceso de Greengrass predeterminada.

Para permitir que el conector interactúe con el daemon de Docker, el usuario de Docker debe pertenecer al grupo de Linux de `docker` del núcleo. El usuario de Docker también debe tener

permisos de escritura en el directorio de `DockerComposeFileDestinationPath`. Aquí es donde el conector almacena el archivo local de `docker-compose.yml` y las credenciales de Docker.

> **Note**
>
> - Le recomendamos que cree un usuario de Linux en lugar de utilizar el `ggc_user` predeterminado. De lo contrario, cualquier función de Lambda del grupo de Greengrass podrá acceder al archivo de Compose y a las credenciales de Docker.
> - Se recomienda evitar la ejecución como raíz a menos que sea absolutamente necesario. Si especifica el usuario raíz, debe permitir que las funciones de Lambda se ejecuten como raíz en AWS IoT Greengrass. Para obtener más información, consulte the section called "Ejecución de una función de Lambda como raíz".

1. Crear el usuario . Puede ejecutar el comando de `useradd` e incluir la opción de `-u` opcional para asignar un ID de usuario. Por ejemplo:

   ```
   sudo useradd -u 1234 user-name
   ```

2. Añada el usuario al grupo de `docker` en el núcleo. Por ejemplo:

   ```
   sudo usermod -aG docker user-name
   ```

   Para obtener más información, como por ejemplo la forma de crear el grupo de `docker`, consulte Manage Docker as a non-root user (Administrar Docker como usuario no raíz) en la documentación de Docker.

3. Otorgue permisos al usuario para que pueda escribir en el directorio especificado para el parámetro de `DockerComposeFileDestinationPath`. Por ejemplo:

   a. Para fijar al usuario como propietario del directorio. En este ejemplo se utiliza el ID de usuario del paso 1.

      ```
      chown 1234 docker-compose-file-destination-path
      ```

   b. Otorgue permisos de lectura y escritura al propietario.

      ```
      chmod 700 docker-compose-file-destination-path
      ```

Para obtener más información, consulte How To Manage File And Folder Permissions In Linux (Cómo administrar los permisos de archivos y carpetas en Linux) en la documentación de Linux Foundation.

c. Si no asignó un ID de usuario al crear el usuario, o si utilizó un usuario existente, ejecute el comando de `id` para buscar el ID de usuario.

```
id -u user-name
```

Utilice el ID de usuario para configurar el parámetro de `DockerUserId` para el conector.

Información de uso

Cuando utilice el conector de implementación de Greengrass Docker, debe tener en cuenta la siguiente información de uso específica de la implementación.

- Prefijo fijo para nombres de proyectos. El conector antepone el prefijo de `greengrassdockerapplicationdeployment` a los nombres de los contenedores Docker que inicia. El conector utiliza este prefijo como nombre del proyecto en los comandos de `docker-compose` que ejecuta.
- Comportamiento de registro. El conector escribe información de estado e información de resolución de problemas en un archivo de registro. Puede configurarlo AWS IoT Greengrass para enviar CloudWatch registros a Logs y escribirlos localmente. Para obtener más información, consulte the section called “Registro”. Esta es la ruta al registro local para el conector:

  ```
  /greengrass-root/ggc/var/log/user/region/aws/DockerApplicationDeployment.log
  ```

  Debe tener permisos de root para acceder a los registros locales.
- Actualización de imágenes de Docker. Docker almacena imágenes en la caché en el dispositivo central. Si actualiza una imagen de Docker y desea propagar el cambio al dispositivo central, asegúrese de cambiar la etiqueta de la imagen en el archivo de Compose. Los cambios surten efecto después de que se implemente el grupo de Greengrass.
- Tiempo de espera de 10 minutos para operaciones de limpieza. Cuando el daemon de Greengrass se detiene durante un reinicio, el comando de `docker-compose down` se activa. Todos los contenedores de Docker tienen un máximo de 10 minutos después de que se active `docker-compose down` para realizar cualquier operación de limpieza. Si la limpieza no se completa en 10

minutos, deberá limpiar manualmente los contenedores restantes. Para obtener más información, consulte docker rm en la documentación de la CLI de Docker.

- Ejecución de comandos de Docker. Para solucionar problemas, puede ejecutar comandos de Docker en una ventana de terminal del dispositivo central. Por ejemplo, ejecute el siguiente comando para ver los contenedores Docker iniciados por el conector:

  ```
  docker ps --filter name="greengrassdockerapplicationdeployment"
  ```

- ID de recurso reservado. El conector utiliza el ID de `DOCKER_DEPLOYER_SECRET_RESOURCE_RESERVED_ID_`*`index`* para los recursos de Greengrass que crea en el grupo de Greengrass. Los ID de recurso deben ser únicos en el grupo, así que no asigne un ID de recurso que pueda entrar en conflicto con este ID de recurso reservado.
- Modo sin conexión. Si establece el parámetro de configuración *`DockerOfflineMode`* en *`True`*, el conector Docker puede funcionar en modo fuera de línea. Esto puede ocurrir cuando la implementación de un grupo de Greengrass se reinicia mientras el dispositivo principal está fuera de línea y el conector no puede establecer una conexión con Amazon S3 o Amazon ECR para recuperar el archivo de Docker Compose.

  Con el modo sin conexión activado, el conector intenta descargar el archivo de Compose y ejecutar los comandos `docker login` como lo haría en un reinicio normal. Si estos intentos fallan, el conector busca un archivo de Compose almacenado localmente en la carpeta que se especificó mediante el parámetro `DockerComposeFileDestinationPath`. Si existe un archivo Compose local, el conector sigue la secuencia normal de los comandos `docker-compose` y extrae imágenes locales. Si el archivo de Compose o las imágenes locales no están presentes, se produce un error en el conector. El comportamiento de los parámetros `ForceDeploy` y `StopContainersOnNewDeployment` sigue siendo el mismo en el modo sin conexión.

Comunicación con contenedores Docker

AWS IoT Greengrass admite los siguientes canales de comunicación entre componentes Greengrass y contenedores Docker:

- Las funciones de Lambda pueden usar API REST para comunicarse con procesos en contenedores de Docker. Puede configurar un servidor en un contenedor de Docker que abre un puerto. Las funciones de Lambda se pueden comunicar con el contenedor de este puerto.

- Los procesos de los contenedores Docker pueden intercambiar mensajes MQTT a través del bróker de mensajería local de Greengrass. Puede configurar el contenedor de Docker como un dispositivo de cliente en el grupo de Greengrass y, a continuación, crear suscripciones para permitir que el contenedor se comunique con funciones, dispositivos y otros conectores de en el grupo, o con AWS IoT y el servicio de sombra local. Para obtener más información, consulte the section called “Configurar la comunicación MQTT con contenedores Docker”.
- Las funciones de Lambda pueden actualizar un archivo compartido para pasar información a contenedores de Docker. Puede utilizar el archivo de Compose para montar un subconjunto de la ruta del archivo compartido en un contenedor Docker.

Configurar la comunicación MQTT con contenedores Docker

Puede configurar un contenedor de Docker como un dispositivo de cliente y añadirlo a un grupo de Greengrass. A continuación, puede crear suscripciones que permitan la comunicación MQTT entre el contenedor Docker y los componentes Greengrass o AWS IoT. En el siguiente procedimiento, va a crear una suscripción que permite que el dispositivo del contenedor Docker reciba mensajes de actualización de sombra desde el servicio de sombra local. Puede seguir este patrón para crear otras suscripciones.

> **Note**
>
> Suponemos que ya ha creado un grupo de Greengrass y un núcleo de Greengrass (versión 1.10 o posterior) en este procedimiento. Para obtener más información acerca de la creación de núcleos y grupos, consulte Empezando con AWS IoT Greengrass.

Para configurar un contenedor de Docker como un dispositivo Greengrass y añadirlo a un grupo de Greengrass

1. Cree una carpeta en el dispositivo central para almacenar los certificados y claves utilizados para autenticar el dispositivo Greengrass.

   La ruta del archivo debe montarse en el contenedor Docker que desee iniciar. El siguiente fragmento muestra cómo montar una ruta de archivo en el archivo de Compose. En este ejemplo, *`path-to-device-certs`*representa la carpeta que creó en este paso.

   ```
   version: '3.3'
   services:
   ```

```
  myService:
    image: user-name/repo:image-tag
    volumes:
      - /path-to-device-certs/:/path-accessible-in-container
```

2. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
3. Seleccione el grupo de destino.
4. En la página de configuración del grupo, seleccione Dispositivos cliente y, a continuación, seleccione Asociar.
5. En el modal Asociar un dispositivo cliente a este grupo, seleccione Crear AWS IoTobjeto nuevo .

   La página Crear objetos se abre en una pestaña nueva.
6. En la página Crea objetos, elija Crear un solo objeto, y luego seleccione Siguiente.
7. En la página Especificar las propiedades del objeto, introduce un nombre para el dispositivo y, a continuación, seleccione Siguiente.
8. En la página Configurar el certificado del dispositivo, seleccione Siguiente.
9. En la página Adjuntar políticas al certificado, realice uno de los siguientes procedimientos:
   - Seleccione una política existente que conceda los permisos que requieren los dispositivos clientes y, a continuación, seleccione Crear objeto.

     Se abre un modal en el que puede descargar los certificados y las claves que el dispositivo utiliza para conectarse al Nube de AWS y al núcleo.
   - Cree y adjunte una nueva política que conceda permisos al dispositivo cliente. Haga lo siguiente:

     a. Elija Crear política.

        La página Create policy (Crear política) se abre en una pestaña nueva.

     b. En la página Create policy (Crear política), haga lo siguiente:

        i. En Nombre de la política, introduzca un nombre que describa la política, como **`GreengrassV1ClientDevicePolicy`**.

        ii. En la pestaña Declaraciones de política, en Documento de política, seleccione JSON.

        iii. Ingrese el siguiente documento de política. Esta política permite que el dispositivo cliente descubra los núcleos de Greengrass y comunique todos los temas MQTT.

Para obtener información acerca de cómo restringir el acceso a esta política, consulte Autenticación y autorización de dispositivos en AWS IoT Greengrass.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. Elija Create (Crear) para crear la política.

c. Vuelva a la pestaña del navegador con la página Adjuntar políticas al certificado abierta. Haga lo siguiente:

i. En la lista de Políticas, seleccione la política que ha creado, como por ejemplo GreengrassV1ClientDevicePolicy.

Si no se puede ver la política, seleccione el botón de actualizar.

ii. Elija Crear objeto.

Se abre un modal en el que puede descargar los certificados y las claves que el dispositivo utiliza para conectarse al Nube de AWS y al núcleo.

10. En el modal Descargar certificados y claves, descargue los certificados del dispositivo.

    > **Important**
    >
    > Descargue los recursos de seguridad antes de elegir Listo.

    Haga lo siguiente:

    a. Para el certificado del dispositivo, seleccione Descargar para descargar el certificado del dispositivo.

    b. En Archivo de clave pública, seleccione Descargar para descargar la clave pública del certificado.

    c. En Archivo de clave privada, seleccione Descargar para descargar el archivo de clave privada del certificado.

    d. Revise la Autenticación de servidor en la Guía del desarrollador de AWS IoT y seleccione el certificado de CA raíz adecuado. Le recomendamos que utilice los puntos de conexión de Amazon Trust Services (ATS) y los certificados de CA raíz de ATS. En Certificados de CA raíz, seleccione Descargar para obtener un certificado de CA raíz.

    e. Seleccione Listo.

    Tome nota del identificador del certificado que comparten los nombres de archivo del certificado y las claves del dispositivo. Lo necesitará más adelante.

11. Copie los certificados y las claves en la carpeta que ha creado en el paso 1.

A continuación, cree una suscripción en el grupo. En este ejemplo, crear una suscripción permite que el dispositivo del contenedor Docker reciba mensajes MQTT del servicio de sombra local.

> **Note**
>
> El tamaño máximo de un documento sombra es de 8 kB. Para obtener más información, consulte Cuotas de AWS IoT en la Guía para desarrolladores de AWS IoT.

Para crear una suscripción que permita al dispositivo del contenedor Docker recibir mensajes MQTT del servicio de sombra local

1. En la página de configuración de grupo, elija Suscripciones y, a continuación, elija Añadir suscripción.
2. En la página Select your source and target, configure el origen y el destino, de la siguiente manera:

   a. Para Select a source (Seleccionar un origen): elija Services (Servicios) y, a continuación, Local Shadow Service (Servicio de sombra local).

   b. En Select a target (Seleccionar un destino), seleccione Devices (Dispositivos), y, a continuación, elija su dispositivo.

   c. Elija Siguiente.

   d. En la página Filtrar los datos por tema, en el campo Filtro por tema, escriba **`$aws/things/`*`MyDockerDevice`*`/shadow/update/accepted`** y, a continuación, seleccione Siguiente. *`MyDockerDevice`*Reemplácelo por el nombre del dispositivo que creó anteriormente.

   e. Seleccione Finalizar.

Incluya el siguiente fragmento de código en la imagen de Docker a la que haga referencia en su archivo de Compose. Este es el código del dispositivo Greengrass. Además, añada el código en el contenedor Docker que inicia el dispositivo Greengrass dentro del contenedor. Se puede ejecutar como un proceso independiente en la imagen o en una cadena independiente.

```
import os
import sys
import time
import uuid

from AWSIoTPythonSDK.core.greengrass.discovery.providers import DiscoveryInfoProvider
from AWSIoTPythonSDK.exception.AWSIoTExceptions import DiscoveryInvalidRequestException
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Replace thingName with the name you registered for the Docker device.
thingName = "MyDockerDevice"
clientId = thingName

# Replace host with the IoT endpoint for your &AWS-account;.
```

```
host = "myPrefix.iot.region.amazonaws.com"

# Replace topic with the topic where the Docker container subscribes.
topic = "$aws/things/MyDockerDevice/shadow/update/accepted"

# Replace these paths based on the download location of the certificates for the Docker
 container.
rootCAPath = "/path-accessible-in-container/AmazonRootCA1.pem"
certificatePath = "/path-accessible-in-container/certId-certificate.pem.crt"
privateKeyPath = "/path-accessible-in-container/certId-private.pem.key"


# Discover Greengrass cores.
discoveryInfoProvider = DiscoveryInfoProvider()
discoveryInfoProvider.configureEndpoint(host)
discoveryInfoProvider.configureCredentials(rootCAPath, certificatePath, privateKeyPath)
discoveryInfoProvider.configureTimeout(10)  # 10 seconds.

GROUP_CA_PATH = "./groupCA/"
MQTT_QOS = 1

discovered = False
groupCA = None
coreInfo = None

try:
    # Get discovery info from AWS IoT.
    discoveryInfo = discoveryInfoProvider.discover(thingName)
    caList = discoveryInfo.getAllCas()
    coreList = discoveryInfo.getAllCores()

    # Use first discovery result.
    groupId, ca = caList[0]
    coreInfo = coreList[0]

    # Save the group CA to a local file.
    groupCA = GROUP_CA_PATH + groupId + "_CA_" + str(uuid.uuid4()) + ".crt"
    if not os.path.exists(GROUP_CA_PATH):
        os.makedirs(GROUP_CA_PATH)
    groupCAFile = open(groupCA, "w")
    groupCAFile.write(ca)
    groupCAFile.close()
    discovered = True
except DiscoveryInvalidRequestException as e:
```

```
    print("Invalid discovery request detected!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")
except BaseException as e:
    print("Error in discovery!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
myAWSIoTMQTTClient.configureCredentials(groupCA, privateKeyPath, certificatePath)


# Try to connect to the Greengrass core.
connected = False
for connectivityInfo in coreInfo.connectivityInfoList:
    currentHost = connectivityInfo.host
    currentPort = connectivityInfo.port
    myAWSIoTMQTTClient.configureEndpoint(currentHost, currentPort)
    try:
        myAWSIoTMQTTClient.connect()
        connected = True
    except BaseException as e:
        print("Error in connect!")
        print("Type: %s" % str(type(e)))
        print("Error message: %s" % str(e))
    if connected:
        break

if not connected:
    print("Cannot connect to core %s. Exiting..." % coreInfo.coreThingArn)
    sys.exit(-2)

# Handle the MQTT message received from GGShadowService.
def customCallback(client, userdata, message):
    print("Received an MQTT message")
    print(message)

# Subscribe to the MQTT topic.
myAWSIoTMQTTClient.subscribe(topic, MQTT_QOS, customCallback)

# Keep the process alive to listen for messages.
while True:
```

```
    time.sleep(1)
```

Notas de seguridad

Cuando utilice el conector de Greengrass Docker, tenga en cuenta las siguientes consideraciones de seguridad.

Almacenamiento local del archivo de Docker Compose

El conector guarda una copia del archivo de Compose en el directorio especificado con el parámetro de `DockerComposeFileDestinationPath`.

Es su responsabilidad proteger este directorio. Debe utilizar los permisos del sistema de archivos para restringir el acceso al directorio.

Almacenamiento local de las credenciales de Docker

Si las imágenes de Docker se guardan en repositorios privados, el conector almacenará las credenciales de Docker en el directorio especificado con el parámetro `DockerComposeFileDestinationPath`.

Es su responsabilidad proteger estas credenciales. Por ejemplo, debe usar credencial-helper en el dispositivo central cuando instale Docker Engine.

Instalar Docker Engine desde una fuente de confianza

Es su responsabilidad instalar Docker Engine desde una fuente de confianza. Este conector utiliza el daemon de Docker del dispositivo central para acceder a sus recursos de Docker y administrar los contenedores de Docker.

Alcance de los permisos de rol del grupo de Greengrass

Todos los conectores y funciones de Lambda del grupo de Greengrass pueden asumir los permisos que añada en el rol del grupo de Greengrass. Este conector requiere acceso al archivo de Docker Compose almacenado en un bucket S3. También requiere acceso a su token de autorización de Amazon ECR si sus imágenes de Docker se almacenan en un repositorio privado en Amazon ECR.

Licencias

El conector de Greengrass Docker incluye las siguientes licencias y software de terceros:

- AWS SDK for Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registros de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
7	Se agregó `DockerOfflineMode` para usar un archivo de Docker Compose existente cuando AWS IoT Greengrass se inicia sin conexión. Se implementaron reintentos para el comando `docker login`. Soporte para UID de 32 bits.
6	Se agregó `StopContainersOnNewDeployment` para anular la limpieza del contenedor cuando se realiza una nueva implementación o se detiene el GGC. Mecanismos de apagado y puesta en marcha más seguros. Corrección de un error de validación de YAML.
5	Las imágenes se extraen antes de ejecutar `docker-compose down` .
4	Se agregó pull-before-up un comportamiento para actualizar las imágenes de Docker.
3	Se ha corregido un problema con la búsqueda de variables de entorno.
2	Se ha agregado el parámetro `ForceDeploy` .

Versión	Cambios
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”

Conector de IoT Analytics

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

El conector IoT Analytics envía datos del dispositivo local a AWS IoT Analytics. Puede utilizar este conector como una ubicación central para recopilar datos de sensores en el dispositivo del núcleo de Greengrass y desde dispositivos cliente conectados. El conector envía los datos a los canales de AWS IoT Analytics en la Cuenta de AWS y región actual. Puede enviar datos a un canal de destino predeterminado y a canales especificados dinámicamente.

> **Note**
>
> AWS IoT Analytics es un servicio completamente administrado que le permite recopilar, almacenar, procesar y consultar datos de IoT. En AWS IoT Analytics, los datos se pueden seguir analizando y procesando. Por ejemplo, se puede utilizar para entrenar modelos de ML para monitorizar el estado de la máquina o para probar nuevas estrategias de modelado.

Para obtener más información, consulte ¿Qué es AWS IoT Analytics? en la Guía del usuario de AWS IoT Analytics.

El conector acepta datos formateados y sin formatear en temas de MQTT de entrada. Admite dos temas predefinidos donde el canal de destino se especifica en línea. También puede recibir mensajes en temas definidos por el cliente que se configuran en suscripciones. Esto se puede utilizar para dirigir mensajes desde dispositivos cliente que se publican a temas fijos o para gestionar datos sin estructurar o dependientes de la pila desde dispositivos con recursos limitados.

Este conector utiliza la API `BatchPutMessage` para enviar datos (como una cadena JSON o con codificación base64) al canal de destino. El conector puede procesar datos sin procesar en un formato que se adapte a los requisitos de la API. El conector almacena en búfer mensajes de entrada en las colas por canal y procesa los lotes de forma asíncrona. Proporciona parámetros que le permiten controlar el comportamiento de creación de colas y procesamiento por lotes y restringir el consumo de memoria. Por ejemplo, puede configurar el tamaño de cola máximo, el intervalo por lotes, el tamaño de memoria y el número de canales activos.

Este conector tiene las siguientes versiones.

Versión	ARN
4	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/4`
3	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/3`
2.	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/2`
1	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3 - 4

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

  > **Note**
  >
  > Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Este conector solo puede utilizarse en las regiones de Amazon Web Services en las que se admiten tanto AWS IoT Greengrass como AWS IoT Analytics.
- Se crean y configuran todas las entidades y flujos de trabajo de AWS IoT Analytics relacionados. Las entidades incluyen canales, canalizaciones, almacenes de datos y conjuntos de datos. Para obtener más información, consulte los procedimientos de la AWS CLI o de la consola en la Guía de usuario de AWS IoT Analytics.

  > **Note**
  >
  > Los canales de AWS IoT Analytics de destino deben usar la misma cuenta y deben estar en la misma Región de AWS que este conector.

- El rol del grupo de Greengrass configurado para permitir la acción `iotanalytics:BatchPutMessage` en los canales de destino, tal y como se muestra en la siguiente política de IAM de ejemplo. Los canales deben estar en la Cuenta de AWS y región actual.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
            ]
        }
    ]
}
```

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Versions 1 - 2

- Software AWS IoT Greengrass Core versión 1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Este conector solo puede utilizarse en las regiones de Amazon Web Services en las que se admiten tanto AWS IoT Greengrass como AWS IoT Analytics.
- Se crean y configuran todas las entidades y flujos de trabajo de AWS IoT Analytics relacionados. Las entidades incluyen canales, canalizaciones, almacenes de datos y conjuntos de datos. Para obtener más información, consulte los procedimientos de la AWS CLI o de la consola en la Guía de usuario de AWS IoT Analytics.

> **Note**
>
> Los canales de AWS IoT Analytics de destino deben usar la misma cuenta y deben estar en la misma Región de AWS que este conector.

- El rol del grupo de Greengrass configurado para permitir la acción `iotanalytics:BatchPutMessage` en los canales de destino, tal y como se muestra en la siguiente política de IAM de ejemplo. Los canales deben estar en la Cuenta de AWS y región actual.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
            ]
        }
    ]
}
```

  Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Parámetros

`MemorySize`

La cantidad de memoria (en KB) para asignar a este conector.

Nombre que mostrar en la consola AWS IoT: Tamaño de memoria

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`PublishRegion`

La Región de AWS en la que se crean sus canales de AWS IoT Analytics. Use la misma región que el conector.

> **Note**
>
> También debe coincidir con la región para los canales que se especifican en el rol de grupo.

Nombre para mostrar en la consola AWS IoT: Publicar región

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^$|([a-z]{2}-[a-z]+-\\d{1})`

`PublishInterval`

El intervalo (en segundos) para publicar un lote de datos recibidos en AWS IoT Analytics.

Nombre para mostrar en la consola AWS IoT: Publicar intervalo

Obligatorio: `false`

Escriba: `string`

Valor predeterminado: `1`

Patrón válido: `$|^[0-9]+$`

`IotAnalyticsMaxActiveChannels`

El número máximo de canales de AWS IoT Analytics que el conector vigila de forma activa. Debe ser mayor que cero e igual o inferior al número de canales en los que espera que el conector publique en un momento determinado.

Puede utilizar este parámetro para restringir el consumo de memoria limitando el número total de colas que el conector puede administrar en un momento dado. Una cola se elimina cuando se han enviado todos los mensajes de colas.

Nombre para mostrar en la consola AWS IoT: Número máximo de canales activos

Obligatorio: `false`

Escriba: `string`

Valor predeterminado: `50`

Patrón válido: `^$|^[1-9][0-9]*$`

`IotAnalyticsQueueDropBehavior`

El comportamiento para eliminar mensajes de una cola de canales cuando la cola está llena.

Nombre para mostrar en la consola AWS IoT: Comportamiento de eliminación de cola

Obligatorio: `false`

Escriba: `string`

Valores válidos: `DROP_NEWEST` o `DROP_OLDEST`

Valor predeterminado: `DROP_NEWEST`

Patrón válido: `^DROP_NEWEST$|^DROP_OLDEST$`

`IotAnalyticsQueueSizePerChannel`

Número máximo de mensajes que conservar en memoria (por canal) antes de que los mensajes se envíen o se eliminen. Debe ser mayor que 0.

Nombre para mostrar en la consola AWS IoT: Tamaño máximo de cola por canal

Obligatorio: `false`

Escriba: `string`

Valor predeterminado: `2048`

Patrón válido: `^$|^[1-9][0-9]*$`

`IotAnalyticsBatchSizePerChannel`

El número máximo de mensajes que enviar a un canal de AWS IoT Analytics en una solicitud por lotes. Debe ser mayor que 0.

Nombre para mostrar en la consola AWS IoT: Número máximo de mensajes por lote por canal

Obligatorio: `false`

Escriba: `string`

Valor predeterminado: 5

Patrón válido: `^$|^[1-9][0-9]*$`

`IotAnalyticsDefaultChannelName`

El nombre del canal de AWS IoT Analytics que este conector utiliza para mensajes que se envían a un tema de entrada definido por el cliente.

Nombre para mostrar en la consola AWS IoT: Nombre de canal predeterminado

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^[a-zA-Z0-9_]$`

`IsolationMode`

El modo de creación de contenedores para este conector. El valor predeterminado es `GreengrassContainer`, lo que significa que el conector se ejecuta en un entorno de tiempo de ejecución aislado dentro del contenedor de AWS IoT Greengrass.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo no se aplica a los conectores.

Nombre para mostrar en la consola AWS IoT: Modo de aislamiento de contenedores

Obligatorio: `false`

Escriba: `string`

Valores válidos: `GreengrassContainer` o `NoContainer`

Patrón válido: `^NoContainer$|^GreengrassContainer$`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector de IoT Analytics.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyIoTAnalyticsApplication",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTAnalytics/
versions/3",
            "Parameters": {
                "MemorySize": "65535",
                "PublishRegion": "us-west-1",
                "PublishInterval": "2",
                "IotAnalyticsMaxActiveChannels": "25",
                "IotAnalyticsQueueDropBehavior": "DROP_OLDEST",
                "IotAnalyticsQueueSizePerChannel": "1028",
                "IotAnalyticsBatchSizePerChannel": "5",
                "IotAnalyticsDefaultChannelName": "my_channel"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de Conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta datos sobre temas de MQTT predefinidos y definidos por el cliente. Los publicadores pueden ser dispositivos cliente, funciones de Lambda u otros conectores.

Temas predefinidos

El conector admite los siguientes dos temas de MQTT estructurados que permiten a los publicadores especificar el nombre de canal insertado.

- Un mensaje con formato en el tema `iotanalytics/channels/+/messages/put`. Los datos de IoT en estos mensajes de entrada se deben formatear como cadena JSON o con codificación base64.
- Un mensaje sin formato en el tema `iotanalytics/channels/+/messages/binary/put`. Los mensajes de entrada recibidos en este tema se tratan como datos binarios y pueden contener cualquier tipo de datos.

  Para publicar en temas predefinidos, reemplace el comodín + por el nombre del canal. Por ejemplo:

  ```
  iotanalytics/channels/my_channel/messages/put
  ```

Temas definidos por el cliente

El conector admite la sintaxis de temas #, que le permite aceptar mensajes de entrada en cualquier tema de MQTT que configure en una suscripción. Le recomendamos que especifique una ruta de tema en lugar de utilizar únicamente el comodín # en sus suscripciones. Estos mensajes se envían al canal predeterminado que especifique para el conector.

Los mensajes de entrada en temas definidos por el cliente se tratan como datos binarios. Pueden utilizar cualquier formato de mensaje y pueden contener cualquier tipo de datos. Puede utilizar temas definidos por el cliente para dirigir mensajes desde dispositivos que publican en temas fijos. También puede utilizarlas para aceptar datos de entrada desde dispositivos de cliente que no pueden procesar los datos en un mensaje con formato para enviar al conector.

Para obtener más información acerca de las suscripciones y temas de MQTT, consulte the section called “Entradas y salidas”.

El rol de grupo debe permitir la acción `iotanalytics:BatchPutMessage` en todos los canales de destino. Para obtener más información, consulte the section called “Requisitos”.

Filtro de temas: `iotanalytics/channels/+/messages/put`

Utilice este tema para enviar mensajes con formato al conector y para especificar de forma dinámica un canal de destino. Este tema también le permite especificar un ID que se devuelve en la salida de respuesta. El conector verifica que los ID sean únicos para cada mensaje en la solicitud `BatchPutMessage` saliente que envía a AWS IoT Analytics. Se elimina un mensaje que tiene un ID duplicado.

Los datos de entrada enviados a este tema deben utilizar el siguiente formato de mensaje.

Propiedades de mensajes

`request`

Los datos que enviar al canal especificado.

Obligatorio: `true`

Escriba: `object` que incluye las siguientes propiedades:

`message`

Los datos de dispositivo o sensor como cadena JSON o con codificación en base64.

Obligatorio: `true`

Escriba: `string`

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a una respuesta de salida. Si se especifica, la propiedad `id` en el objeto de respuesta se establece en este valor. Si omite esta propiedad, el conector genera un ID.

Obligatorio: `false`

Escriba: `string`

Patrón válido: `.*`

Ejemplo de entrada

```
{
    "request": {
        "message" : "{\"temp\":23.33}"
    },
    "id" : "req123"
```

```
}
```

Filtro de temas: `iotanalytics/channels/+/messages/binary/put`

Utilice este tema para enviar mensajes sin formato al conector y para especificar de forma dinámica un canal de destino.

Los datos del conector no analizan los mensajes de entrada recibidos en este tema. Los trata como datos binarios. Antes de enviar los mensajes a AWS IoT Analytics, el conector los codifica y formatea de acuerdo con los requisitos de la API `BatchPutMessage`:

- El conector codifica en base64 los datos sin formato e incluye la carga cifrada en una solicitud `BatchPutMessage` saliente.
- El conector genera y asigna un ID a cada mensaje de entrada.

> **Note**
>
> La salida de respuesta del conector no incluye una correlación de ID para estos mensajes de entrada.

Propiedades de mensajes

Ninguno.

Filtro de temas: #

Utilice este tema para enviar cualquier formato de mensaje al canal predeterminado. Esto resulta especialmente útil cuando los dispositivos cliente publican en temas fijos o cuando desea enviar datos al canal predeterminado desde dispositivos de clientes que no pueden procesar los datos en el formato de mensaje admitido del conector.

Defina la sintaxis del tema en la suscripción que crea para conectar este conector al origen de datos. Le recomendamos que especifique una ruta de tema en lugar de utilizar únicamente el comodín # en sus suscripciones.

Los datos del conector no analizan los mensajes que se publican en este tema de entrada. Todos los mensajes de entrada se tratan como datos binarios. Antes de enviar los mensajes a AWS IoT Analytics, el conector los codifica y formatea de acuerdo con los requisitos de la API `BatchPutMessage`:

- El conector codifica en base64 los datos sin formato e incluye la carga cifrada en una solicitud `BatchPutMessage` saliente.

- El conector genera y asigna un ID a cada mensaje de entrada.

> **Note**
>
> La salida de respuesta del conector no incluye una correlación de ID para estos mensajes de entrada.

Propiedades de mensajes

Ninguno.

Datos de salida

Este conector publica información de estado como datos de salida en un tema MQTT. Esta información contiene la respuesta devuelta por AWS IoT Analytics para cada mensaje de entrada que recibe y envía a AWS IoT Analytics.

Filtro de temas en la suscripción

```
iotanalytics/messages/put/status
```

Ejemplo de salida: Correcto

```
{
    "response" : {
        "status" : "success"
    },
    "id" : "req123"
}
```

Ejemplo de salida: Error

```
{
    "response" : {
        "status" : "fail",
        "error" : "ResourceNotFoundException",
        "error_message" : "A resource with the specified name could not be found."
    },
    "id" : "req123"
}
```

> **Note**
>
> Si el conector detecta un error que se puede volver a intentar (por ejemplo, errores de conexión), volverá a intentar la publicación en el siguiente lote. El retroceso exponencial lo gestiona el SDK de AWS. Las solicitudes que fallan con errores que se pueden reintentar se añaden de nuevo a la cola del canal para seguir publicándolos de acuerdo con el parámetro `IotAnalyticsQueueDropBehavior`.

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.
> - Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.

   Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de AWS IoT Greengrass Core para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.

3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el conector y configure sus parámetros.

   c. Agregue suscripciones que permitan al conector recibir datos de entrada y enviar datos de salida en filtros de tema compatibles.

      - Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
      - Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.

4. Implemente el grupo.

5. En la consola de AWS IoT, en la página Prueba, suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

   Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la función envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'iotanalytics/channels/my_channel/messages/put'

def create_request_with_all_fields():
    return  {
        "request": {
            "message" : "{\"temp\":23.33}"
        },
        "id" : "req_123"
```

```
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Límites

Este conector está sujeto a los siguientes límites.

- Todos los límites impuestos por AWS SDK for Python (Boto3) para la acción AWS IoT Analytics `batch_put_message`.
- Todas las cuotas impuestas por la API BatchputMessage AWS IoT Analytics. Para obtener más información, consulte Service Quotas de AWS IoT Analytics en la Referencia general de AWS.
  - 100 000 mensajes por segundo por canal.
  - 100 mensajes por lote.
  - 128 KB por mensaje.

  Esta API utiliza nombres de canal (no ARN de canal), por tanto no se admite el envío de datos a canales entre regiones o entre cuentas.
- Todas las cuotas impuestas por AWS IoT Greengrass Core. Para obtener más información, consulte Service Quotas para el core de AWS IoT Greengrass en la Referencia general de AWS.

  Las siguientes cuotas se podrían aplicar:
  - El tamaño máximo de los mensajes enviados por un dispositivo es 128 KB.
  - El tamaño máximo de cola de mensajes del router principal de Greengrass es 2,5 MB.
  - La longitud máxima de una cadena de tema es 256 bytes de caracteres codificados en UTF-8.

Licencias

El conector de Internet incluye las siguientes licencias y software de terceros:

- AWS SDK for Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
4	Añade el parámetro `IsolationMode` para configurar el modo de creación de contenedores del conector.
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se ha introducido una corrección para reducir el registro excesivo.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores

- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- ¿Qué es AWS IoT Analytics? en la Guía del usuario de AWS IoT Analytics

Conector de adaptador de protocolo IP Ethernet IoT

El conector del adaptador de protocolo IP Ethernet IoT recopila datos de dispositivos locales mediante el protocolo EtherNet/IP. Puede utilizar este conector para recopilar datos de varios servidores y publicarlos en `StreamManager`.

También puede usar este conector con el conector IoT SiteWise y tu puerta de enlace IoT SiteWise. Su puerta de enlace debe proporcionar la configuración del conector. Para obtener más información, consulte Configurar una fuente EtherNet/IP (EIP) en la guía del usuario de IoT SiteWise.

> **Note**
>
> Este conector se ejecuta en modo sin aislamiento de contenedores, por lo que puede implementarlo en un grupo AWS IoT Greengrass que se ejecute en un contenedor de Docker.

Este conector tiene las siguientes versiones.

Versión	ARN
2 (recomendado)	`arn:aws:greengrass:  region::/connectors/IoTEIPProtocolAdaptor/versions/2`
1	`arn:aws:greengrass:  region::/connectors/IoTEIPProtocolAdaptor/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 1 and 2

- Software AWS IoT Greengrass Core versión 1.10.2 o posterior.
- Administrador de secuencias habilitado en el grupo de AWS IoT Greengrass.
- Java 8 instalado en el dispositivo de núcleo y añadido a la variable de entorno PATH.
- Un mínimo de 256 MB de RAM adicionales. Este requisito se suma a los requisitos de memoria AWS IoT Greengrass Core.

> **Note**
>
> Este conector sólo está disponible en las siguientes regiones:
>
> - cn-north-1
> - ap-southeast-1
> - ap-southeast-2
> - eu-central-1
> - eu-west-1
> - us-east-1
> - us-west-2

Parámetros de conector

Este conector admite los siguientes parámetros:

`LocalStoragePath`

El directorio del host de AWS IoT Greengrass en la que el conector de IoT SiteWise puede escribir datos persistentes. El directorio predeterminado es `/var/sitewise`.

Nombre para mostrar en la consola AWS IoT: Ruta de almacenamiento local

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^\s*$|\/.`

`ProtocolAdapterConfiguration`

El conjunto de configuraciones de recopiladores EtherNet/IP desde las que el conector recopila datos o a las que se conecta. Puede ser una lista vacía.

Nombre para mostrar en la consola AWS IoT: Configuración del adaptador de protocolo

Obligatorio: `true`

Tipo: una cadena JSON bien formada que define el conjunto de configuraciones de comentarios admitidas.

A continuación se muestra un ejemplo de una `ProtocolAdapterConfiguration`.

```
{
    "sources": [
        {
            "type": "EIPSource",
            "name": "TestSource",
            "endpoint": {
                "ipAddress": "52.89.2.42",
                "port": 44818
            },
            "destination": {
                "type": "StreamManager",
                "streamName": "MyOutput_Stream",
                "streamBufferSize": 10
            },
            "destinationPathPrefix": "EIPSource_Prefix",
            "propertyGroups": [
                {
                    "name": "DriveTemperatures",
                    "scanMode": {
                        "type": "POLL",
                        "rate": 10000
                    },
                    "tagPathDefinitions": [
                        {
                            "type": "EIPTagPath",
```

```
                            "path": "arrayREAL[0]",
                            "dstDataType": "double"
                        }
                    ]
                }
            ]
        }
    ]
}
```

Ejemplo de creación de conector (AWS CLI)

El siguiente comando CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector Adaptador de protocolo IP Ethernet IoT.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version
'{
    "Connectors": [
        {
            "Id": "MyIoTEIPProtocolConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
IoTEIPProtocolAdaptor/versions/2",
            "Parameters": {
                "ProtocolAdaptorConfiguration": "{ \"sources\": [{ \"type
\": \"EIPSource\", \"name\": \"Source1\", \"endpoint\": { \"ipAddress\":
 \"54.245.77.218\", \"port\": 44818 }, \"destinationPathPrefix\": \"EIPConnector_Prefix
\", \"propertyGroups\": [{ \"name\": \"Values\", \"scanMode\": { \"type\": \"POLL\",
 \"rate\": 2000 }, \"tagPathDefinitions\": [{ \"type\": \"EIPTagPath\", \"path\":
 \"arrayREAL[0]\", \"dstDataType\": \"double\" }]}]}]}",
                "LocalStoragePath": "/var/MyIoTEIPProtocolConnectorState"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

Datos de entrada

Este conector no acepta mensajes MQTT como datos de entrada.

Datos de salida

Este conector publica datos en `StreamManager`. Debe configurar el flujo de mensajes de destino. Los mensajes de salida tienen la siguiente estructura:

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

Licencias

El conector Adaptador de protocolo IP Ethernet IoT incluye el siguiente software/licencias de terceros:

- Cliente EtherNet/IP
- MapDB
- Elsa

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios	Fecha
2	Esta versión contiene correcciones de errores.	23 de diciembre de 2021

Versión	Cambios	Fecha
1	Versión inicial.	15 de diciembre de 2020

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”

SiteWise Conector IoT

El SiteWise conector IoT envía datos de dispositivos y equipos locales a las propiedades de los activosAWS IoT SiteWise. Puede utilizar este conector para recopilar datos de varios servidores OPC-UA y publicarlos en el IoT. SiteWise El conector envía los datos a las propiedades de recurso de la Cuenta de AWS y la región actuales.

> **Note**
>
> SiteWise El IoT es un servicio totalmente gestionado que recopila, procesa y visualiza datos de dispositivos y equipos industriales. Puede configurar las propiedades de recurso que procesan los datos sin procesar enviados desde este conector a las propiedades de medición de sus recursos. Por ejemplo, puede definir una propiedad de transformación que convierta los puntos de datos en temperatura Celsius de un dispositivo a Fahrenheit, o puede definir una propiedad métrica que calcule la temperatura media por hora. Para obtener más información, consulte ¿Qué es AWS IoT SiteWise? en la Guía del usuario de AWS IoT SiteWise.

El conector envía datos al IoT SiteWise con las rutas de flujo de datos OPC-UA enviadas desde los servidores OPC-UA. Por ejemplo, la ruta del flujo de datos `/company/windfarm/3/turbine/7/temperature` podría representar el sensor de temperatura de la turbina n.º 7 en el parque eólico

n.º 3. Si AWS IoT Greengrass del núcelo pierde la conexión a Internet, el conector almacenará los datos en la caché hasta que pueda conectarse correctamente a la Nube de AWS. Puede configurar el tamaño máximo del búfer de disco utilizado para el almacenamiento de datos en la caché. Si el tamaño de la caché excede el tamaño máximo del búfer de disco, el conector descartará los datos más antiguos de la cola.

Después de configurar e implementar el SiteWise conector de IoT, puede agregar una puerta de enlace y fuentes OPC-UA en la consola de IoT SiteWise . Al configurar una fuente en la consola, puede filtrar o prefijar las rutas de flujo de datos OPC-UA enviadas por el conector IoT. SiteWise Para obtener instrucciones sobre cómo terminar de configurar la gateway y los orígenes, consulte Añadir la gateway en la Guía del usuario de AWS IoT SiteWise.

SiteWise El IoT recibe datos únicamente de los flujos de datos que usted ha asignado a las propiedades de medición de los SiteWise activos de IoT. Para asignar secuencias de datos a propiedades de recurso, puede establecer el alias de una propiedad para que sea equivalente a una ruta de flujo de datos OPC-UA. Para obtener más información sobre cómo definir modelos de recursos y crear recursos, consulte Crear modelos de recursos industriales en la Guía del usuario de AWS IoT SiteWise.

> **Notas**
>
> Puedes usar el administrador de transmisiones para cargar datos al IoT SiteWise desde fuentes distintas a los servidores OPC-UA. El administrador de secuencias también ofrece soporte personalizable para la gestión de la persistencia y el ancho de banda. Para obtener más información, consulte Administrar secuencias de datos.
>
> Este conector se ejecuta en modo sin aislamiento de contenedores, por lo que puede implementarlo en un grupo Greengrass que se ejecute en un contenedor de Docker.

Este conector tiene las siguientes versiones.

Versión	ARN
12 (recomendado)	`arn:aws:greengrass:  region::/connectors/IoTSiteWise/versions/ 12`

Versión	ARN
11	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/11
10	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/10
9	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/9
8	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/8
7	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/7
6	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/6
5	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/5
4	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/4
3	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/3

Versión	ARN
2.	`arn:aws:greengrass:  region::/connectors/IoTSiteWise/versions/ 2`
1	`arn:aws:greengrass:  region::/connectors/IoTSiteWise/versions/ 1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 9, 10, 11, and 12

> **Important**
>
> Esta versión introduce nuevos requisitos: Software de AWS IoT Greengrass Core versión 1.10.2 y el administrador de secuencias.

- Software AWS IoT Greengrass Core versión 1.10.2
- Administrador de secuencias habilitado en el grupo de Greengrass.
- Java 8 instalado en el dispositivo de núcleo y añadido a la variable de entorno PATH.
- Este conector solo se puede utilizar en las regiones de Amazon Web Services en las que se admiten AWS IoT Greengrasstanto el IoT como el Internet de SiteWise las Cosas.
- Una política de IAM añadida al rol de grupo de Greengrass. Este rol permite el acceso de grupo de AWS IoT Greengrass a la acción `iotsitewise:BatchPutAssetPropertyValue` en el recurso raíz de destino y sus elementos secundarios, tal y como se muestra en el siguiente ejemplo. Puede eliminarlo `Condition` de la política para permitir que el conector acceda a todos sus SiteWise activos de IoT.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Versions 6, 7, and 8

> **Important**
>
> Esta versión introduce nuevos requisitos: Software de AWS IoT Greengrass Core v1.10.0 y stream manager.

- Software AWS IoT Greengrass Core versión 1.10.0
- Administrador de secuencias habilitado en el grupo de Greengrass.
- Java 8 instalado en el dispositivo de núcleo y añadido a la variable de entorno PATH.
- Este conector solo se puede utilizar en las regiones de Amazon Web Services en las que se admiten AWS IoT Greengrasstanto el IoT como el Internet de SiteWise las Cosas.
- Una política de IAM añadida al rol de grupo de Greengrass. Este rol permite el acceso de grupo de AWS IoT Greengrass a la acción `iotsitewise:BatchPutAssetPropertyValue` en el recurso raíz de destino y sus elementos secundarios, tal y como se muestra en el siguiente ejemplo. Puede eliminarlo `Condition` de la política para permitir que el conector acceda a todos sus SiteWise activos de IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Version 5

- Software AWS IoT Greengrass Core versión 1.9.4.
- Java 8 instalado en el dispositivo de núcleo y añadido a la variable de entorno PATH.
- Este conector solo se puede utilizar en las regiones de Amazon Web Services en las que se admiten AWS IoT Greengrasstanto el IoT como el Internet de SiteWise las Cosas.
- Una política de IAM añadida al rol de grupo de Greengrass. Este rol permite el acceso de grupo de AWS IoT Greengrass a la acción `iotsitewise:BatchPutAssetPropertyValue` en el recurso raíz de destino y sus elementos secundarios, tal y como se muestra en el siguiente ejemplo. Puede eliminarlo `Condition` de la política para permitir que el conector acceda a todos sus SiteWise activos de IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
```

```
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
                }
            }
        }
    ]
}
```

Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Version 4

- Software AWS IoT Greengrass Core versión 1.10.0
- Java 8 instalado en el dispositivo de núcleo y añadido a la variable de entorno PATH.
- Este conector solo se puede utilizar en las regiones de Amazon Web Services en las que se admiten AWS IoT Greengrasstanto el IoT como el Internet de SiteWise las Cosas.
- Una política de IAM añadida al rol de grupo de Greengrass. Este rol permite el acceso de grupo de AWS IoT Greengrass a la acción `iotsitewise:BatchPutAssetPropertyValue` en el recurso raíz de destino y sus elementos secundarios, tal y como se muestra en el siguiente ejemplo. Puede eliminarlo `Condition` de la política para permitir que el conector acceda a todos sus SiteWise activos de IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
```

```
                        "/root node asset ID/*"
                    ]
                }
            }
        }
    ]
}
```

Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Version 3

- Software AWS IoT Greengrass Core versión 1.9.4.
- Java 8 instalado en el dispositivo de núcleo y añadido a la variable de entorno PATH.
- Este conector solo se puede utilizar en las regiones de Amazon Web Services en las que se admiten AWS IoT Greengrasstanto el IoT como el Internet de SiteWise las Cosas.
- Una política de IAM añadida al rol de grupo de Greengrass. Este rol permite el acceso de grupo de AWS IoT Greengrass a la acción `iotsitewise:BatchPutAssetPropertyValue` en el recurso raíz de destino y sus elementos secundarios, tal y como se muestra en el siguiente ejemplo. Puede eliminarlo `Condition` de la política para permitir que el conector acceda a todos sus SiteWise activos de IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
```

```
}
```

Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Versions 1 and 2

- Software AWS IoT Greengrass Core versión 1.9.4.
- Java 8 instalado en el dispositivo de núcleo y añadido a la variable de entorno PATH.
- Este conector solo se puede utilizar en las regiones de Amazon Web Services en las que se admiten AWS IoT Greengrasstanto el IoT como el Internet de SiteWise las Cosas.
- Una política de IAM añadida al rol del grupo de Greengrass que permita el acceso a la acción AWS IoT Core y `iotsitewise:BatchPutAssetPropertyValue` en el recurso raíz de destino y todos sus elementos secundarios, tal y como se muestra en el siguiente ejemplo. Puede eliminarlo `Condition` de la política para permitir que el conector acceda a todos sus SiteWise activos de IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        },
        {
            "Effect": "Allow",
            "Action": [
                 "iot:Connect",
                 "iot:DescribeEndpoint",
                 "iot:Publish",
                 "iot:Receive",
```

```
                "iot:Subscribe"
            ],
            "Resource": "*"
        }
    ]
}
```

Para más información, consulte Adición y eliminación de permisos de identidad de IAM en la Guía del usuario de IAM de .

Parámetros

Versions 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12

`SiteWiseLocalStoragePath`

El directorio del AWS IoT Greengrass host en el que el SiteWise conector de IoT puede escribir datos persistentes. El valor predeterminado es `/var/sitewise`.

Nombre para mostrar en la consola AWS IoT: Ruta de almacenamiento local

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^\s*$|\/.`

`AWSSecretsArnList`

Lista de secretos en AWS Secrets Manager. Cada secreto contiene un par clave-valor de contraseña y nombre de usuario de OPC-UA. Cada secreto debe ser un secreto de tipo par clave-valor.

Nombre que mostrar en la consola AWS IoT: Lista de ARN para los secretos de nombre de usuario/contraseña de OPC-UA

Obligatorio: `false`

Tipo: `JsonArrayOfStrings`

Patrón válido: `\[( ?,? ?\"(arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+\/)*[a-zA-Z0-9\/_+=,.@\\-]+-[a-zA-Z0-9]+)*\")*\]`

`MaximumBufferSize`

El tamaño máximo en GB para el uso SiteWise del disco de IoT. El valor predeterminado es 10 GB.

Nombre para mostrar en la consola AWS IoT: Tamaño máximo del búfer de disco

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^\s*$|[0-9]+`

Version 1

`SiteWiseLocalStoragePath`

El directorio del AWS IoT Greengrass host en el que el SiteWise conector de IoT puede escribir datos persistentes. El valor predeterminado es `/var/sitewise`.

Nombre para mostrar en la consola AWS IoT: Ruta de almacenamiento local

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^\s*$|\/.`

`SiteWiseOpcuaUserIdentityTokenSecretArn`

El secreto en AWS Secrets Manager que contiene el par clave-valor de contraseña y nombre de usuario de OPC-UA. Este secreto debe ser un secreto de tipo par clave-valor.

Nombre para mostrar en la consola AWS IoT: ARN del nombre de usuario/contraseña secreto de OPC-UA

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^$|arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+/)*[a-zA-Z0-9/_+=,.@\\-]+-[a-zA-Z0-9]+`

`SiteWiseOpcuaUserIdentityTokenSecretArn-ResourceId`

El recurso secreto en el grupo de AWS IoT Greengrass que hace referencia a un secreto de nombre de usuario y contraseña de OPC-UA.

Nombre para mostrar en la consola AWS IoT: Recurso secreto de nombre de usuario/contraseña OPC-UA

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^$|.+`

`MaximumBufferSize`

El tamaño máximo en GB para el uso SiteWise del disco de IoT. El valor predeterminado es 10 GB.

Nombre para mostrar en la consola AWS IoT: Tamaño máximo del búfer de disco

Obligatorio: `false`

Tipo: `string`

Patrón válido: `^\s*$|[0-9]+`

Ejemplo de creación de conector (AWS CLI)

El siguiente AWS CLI comando crea una `ConnectorDefinition` con una versión inicial que contiene el SiteWise conector IoT.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyIoTSiteWiseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTSiteWise/
versions/11"
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector no acepta mensajes MQTT como datos de entrada.

Datos de salida

Este conector no publica los mensajes MQTT como datos de salida.

Límites

Este conector está sujeto a los siguientes límites impuestos por el IoT SiteWise, incluidos los siguientes. Para obtener más información, consulte puntos de conexión y cuotas de AWS IoT SiteWise en la Referencia general de AWS.

- Cantidad máxima de puertas de enlace por Cuenta de AWS.
- Cantidad máxima de orígenes OPC-UA por gateway.
- Velocidad máxima de puntos de datos timestamp-quality-value (TQV) almacenados por. Cuenta de AWS
- Tasa máxima de puntos de datos de TQV almacenados por propiedad de recurso.

Licencias

Version 9, 10, 11, and 12

El SiteWise conector IoT incluye el siguiente software o licencia de terceros:

- MapDB
- Elsa
- Eclipse Milo

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Versions 6, 7, and 8

El SiteWise conector IoT incluye el siguiente software o licencia de terceros:

- Milo / EDL 1.0

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Versions 1, 2, 3, 4, and 5

El SiteWise conector IoT incluye el siguiente software o licencia de terceros:

- Milo / EDL 1.0
- Chronicle-Queue / Apache License 2.0

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registros de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios	Date
12	• Esta versión contiene correcciones de errores.	22 de diciembre de 2021
11	• Soporte para cadenas que contienen caracteres ocultos o no imprimibles. Los caracteres ocultos y no imprimibles se eliminan automáticamente antes de enviar las cadenas a la Nube de AWS. • Se ha corregido un problema que provocaba que la SiteWise puerta de	24 de marzo de 2021

Versión	Cambios	Date
	enlace de IoT reintentara de forma infinita las solicitudes no válidas. • Se ha corregido un problema que provocaba que un punto de control se dañara cuando la SiteWise puerta de enlace de IoT estaba conectada a una fuente de datos de alta frecuencia. • Se han mejorado los mensajes de error para ayudar a solucionar los problemas de configuración de la puerta de enlace.	
10	`StreamManager` configura do para mejorar el manejo cuando la conexión de origen se pierde y se restablece. Esta versión también acepta valores OPC-UA con un signo `ServerTimestamp` cuando no hay un `SourceTim estamp` disponible.	22 de enero de 2021

Versión	Cambios	Date
9	Se ha lanzado la compatibilidad con los destinos personalizados de la transmisión `StreamManager` de Greengrass, la banda muerta OPC-UA, el modo de escaneado personalizado y la velocidad de escaneado personalizada. También incluye un rendimiento mejorado durante las actualizaciones de configuración realizadas desde la SiteWise puerta de enlace de IoT.	15 de diciembre de 2020
8	Estabilidad mejorada cuando el conector experimenta una conectividad de red intermitente.	19 de noviembre de 2020
7	Se ha corregido un problema con las métricas de la puerta de enlace.	14 de agosto de 2020
6	Se agregó soporte para CloudWatch métricas y detección automática de nuevas etiquetas OPC-UA. Esta versión requiere el administrador de secuencias y el software AWS IoT Greengrass Core v1.10.0 o superior.	29 de abril de 2020

Versión	Cambios	Date
5	Corrigió un problema de compatibilidad con el software de AWS IoT Greengrass Core v1.9.4.	12 de febrero de 2020
4	Se ha corregido un problema con la reconexión del servidor OPC-UA.	7 de febrero de 2020
3	Requisito de permisos `iot:*` eliminado.	17 de diciembre de 2019
2	Se ha añadido compatibi lidad con varios recursos de secretos de OPC-UA.	10 de diciembre de 2019
1	Versión inicial.	2 de diciembre de 2019

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- Consulte los siguientes temas en la Guía del usuario de AWS IoT SiteWise:
  - ¿Qué es AWS IoT SiteWise?
  - Uso de una puerta de enlace
  - Métricas de Gateway CloudWatch
  - Solución de problemas de una SiteWise puerta de enlace de IoT

Kinesis Firehose

El conector Kinesis Firehose publica los datos a través de una transmisión de entrega de Amazon Data Firehose a destinos como Amazon S3, Amazon Redshift o Amazon Service. OpenSearch

Este conector es un productor de datos de un flujo de entrega de Kinesis. Recibe los datos de entrada en un tema de MQTT y envía los datos a una transmisión de entrega especificada. La transmisión de entrega, a continuación, envía el registro de datos al destino configurado (por ejemplo, un bucket de S3).

Este conector tiene las siguientes versiones.

Versión	ARN
5	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/5`
4	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/4`
3	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/3`
2.	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/2`
1	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 4 - 5

- AWS IoT Greengrass Software principal, versión 1.9.3 o posterior.
- Se necesita tener la versión 3.7 o 3.8 de Python instalada en el dispositivo principal y añadido a la variable de entorno PATH.

> **Note**
>
> Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Una secuencia de entrega de Kinesis configurada. Para obtener más información, consulte Creación de una transmisión de entrega de Amazon Data Firehose en la Guía para desarrolladores de Amazon Kinesis Firehose.
- El rol del grupo de Greengrass configurado para permitir las acciones `firehose:PutRecord` y `firehose:PutRecordBatch` en la transmisión de entrega de destino, tal y como se muestra en la siguiente política de IAM de ejemplo.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
```

```
}
```

Este conector le permite anular dinámicamente la transmisión de entrega predeterminada en la carga de mensajes de entrada. Si su implementación utiliza esta característica, la política de IAM debe incluir todas las transmisiones de destino como recursos. Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *)

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Versions 2 - 3

- AWS IoT Greengrass Software básico v1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Una secuencia de entrega de Kinesis configurada. Para obtener más información, consulte Creación de una transmisión de entrega de Amazon Data Firehose en la Guía para desarrolladores de Amazon Kinesis Firehose.
- El rol del grupo de Greengrass configurado para permitir las acciones `firehose:PutRecord` y `firehose:PutRecordBatch` en la transmisión de entrega de destino, tal y como se muestra en la siguiente política de IAM de ejemplo.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
```

```
    ]
}
```

Este conector le permite anular dinámicamente la transmisión de entrega predeterminada en la carga de mensajes de entrada. Si su implementación utiliza esta característica, la política de IAM debe incluir todas las transmisiones de destino como recursos. Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *)

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Version 1

- AWS IoT Greengrass Software básico v1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Una secuencia de entrega de Kinesis configurada. Para obtener más información, consulte Creación de una transmisión de entrega de Amazon Data Firehose en la Guía para desarrolladores de Amazon Kinesis Firehose.
- El rol del grupo de Greengrass configurado para permitir la acción `firehose:PutRecord` en la transmisión de entrega de destino, tal y como se muestra en la siguiente política de IAM de ejemplo.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
```

```
    ]
}
```

Este conector le permite anular dinámicamente la transmisión de entrega predeterminada en la carga de mensajes de entrada. Si su implementación utiliza esta característica, la política de IAM debe incluir todas las transmisiones de destino como recursos. Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *)

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

Versions 5

`DefaultDeliveryStreamArn`

El ARN del flujo de entrega predeterminado de Firehose al que se van a enviar los datos. La transmisión de destino se puede anular mediante la propiedad `delivery_stream_arn` en la carga de mensajes de entrada.

> **Note**
>
> El rol de grupo debe permitir las acciones adecuadas en todas las transmisiones de entrega de destino. Para obtener más información, consulte the section called “Requisitos”.

Nombre para mostrar en la AWS IoT consola: ARN del flujo de entrega predeterminado

Obligatorio: `true`

Tipo: `string`

Patrón válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

El número máximo de registros que mantener en memoria antes de rechazar los nuevos registros de la misma secuencia de entrega. El valor mínimo es 2000.

Nombre para mostrar en la AWS IoT consola: número máximo de registros para almacenar en búfer (por transmisión)

Obligatorio: `true`

Tipo: `string`

Patrón válido: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

La cantidad de memoria (en KB) para asignar a este conector.

Nombre para mostrar en la AWS IoT consola: tamaño de memoria

Obligatorio: `true`

Tipo: `string`

Patrón válido: `^[0-9]+$`

`PublishInterval`

El intervalo (en segundos) para publicar registros en Firehose. Para desactivar el procesamiento por lotes, establezca este valor en 0.

Nombre para mostrar en la AWS IoT consola: intervalo de publicación

Obligatorio: `true`

Tipo: `string`

Valores válidos: `0 - 900`

Patrón válido: `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

`IsolationMode`

El modo de creación de contenedores para este conector. El valor predeterminado es`GreengrassContainer`, lo que significa que el conector se ejecuta en un entorno de ejecución aislado dentro del AWS IoT Greengrass contenedor.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo no se aplica a los conectores.

Nombre para mostrar en la AWS IoT consola: modo de aislamiento del contenedor

Obligatorio: `false`

Tipo: `string`

Valores válidos: `GreengrassContainer` o `NoContainer`

Patrón válido: `^NoContainer$|^GreengrassContainer$`

Versions 2 - 4

`DefaultDeliveryStreamArn`

El ARN del flujo de entrega predeterminado de Firehose al que se van a enviar los datos. La transmisión de destino se puede anular mediante la propiedad `delivery_stream_arn` en la carga de mensajes de entrada.

> **Note**
>
> El rol de grupo debe permitir las acciones adecuadas en todas las transmisiones de entrega de destino. Para obtener más información, consulte the section called “Requisitos”.

Nombre para mostrar en la AWS IoT consola: ARN del flujo de entrega predeterminado

Obligatorio: `true`

Tipo: `string`

Patrón válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

El número máximo de registros que mantener en memoria antes de rechazar los nuevos registros de la misma secuencia de entrega. El valor mínimo es 2000.

Nombre para mostrar en la AWS IoT consola: número máximo de registros para almacenar en búfer (por transmisión)

Obligatorio: `true`

Tipo: `string`

Patrón válido: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

La cantidad de memoria (en KB) para asignar a este conector.

Nombre para mostrar en la AWS IoT consola: tamaño de memoria

Obligatorio: `true`

Tipo: `string`

Patrón válido: `^[0-9]+$`

`PublishInterval`

El intervalo (en segundos) para publicar registros en Firehose. Para desactivar el procesamiento por lotes, establezca este valor en 0.

Nombre para mostrar en la AWS IoT consola: intervalo de publicación

Obligatorio: `true`

Tipo: `string`

Valores válidos: `0 - 900`

Patrón válido: `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

Version 1

`DefaultDeliveryStreamArn`

El ARN del flujo de entrega predeterminado de Firehose al que se van a enviar los datos. La transmisión de destino se puede anular mediante la propiedad `delivery_stream_arn` en la carga de mensajes de entrada.

> **Note**
>
> El rol de grupo debe permitir las acciones adecuadas en todas las transmisiones de entrega de destino. Para obtener más información, consulte the section called "Requisitos".

Nombre para mostrar en la AWS IoT consola: ARN del flujo de entrega predeterminado

Obligatorio: `true`

Tipo: `string`

Patrón válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

Example

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyKinesisFirehoseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/KinesisFirehose/
versions/5",
            "Parameters": {
                "DefaultDeliveryStreamArn": "arn:aws:firehose:region:account-
id:deliverystream/stream-name",
```

```
                "DeliveryStreamQueueSize": "5000",
                "MemorySize": "65535",
                "PublishInterval": "10",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

En la AWS IoT Greengrass consola, puede añadir un conector desde la página de conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta el contenido de transmisión en los temas de MQTT y, a continuación, envía el contenido a la transmisión de entrega de destino. Acepta dos tipos de datos de entrada:

- Datos JSON en el tema `kinesisfirehose/message`.
- Datos binarios en el tema `kinesisfirehose/message/binary/#`.

Versions 2 - 5

Filtro de temas: `kinesisfirehose/message`

Consulte este tema para enviar un mensaje que contenga datos JSON.

Propiedades de mensajes

`request`

Los datos para enviar a la transmisión de entrega y a la transmisión de entrega de destino, si no es la transmisión predeterminada.

Obligatorio: `true`

Escriba: `object` que incluye las siguientes propiedades:

`data`

Los datos que enviar a la transmisión de entrega.

Obligatorio: `true`

Tipo: `string`

`delivery_stream_arn`

El ARN de la transmisión de entrega de Kinesis de destino. Incluya esta propiedad para anular la transmisión de entrega predeterminada.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a una respuesta de salida. Si se especifica, la propiedad `id` en el objeto de respuesta se establece en este valor. Si no utiliza esta característica, puede omitir esta propiedad o especificar una cadena vacía.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `.*`

Ejemplo de entrada

```
{
     "request": {
        "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
        "data": "Data to send to the delivery stream."
     },
     "id": "request123"
}
```

Filtro de temas: `kinesisfirehose/message/binary/#`

Consulte este tema para enviar un mensaje que contenga datos binarios. El conector no analiza los datos binarios. Los datos se transmiten tal cual.

Para asignar la solicitud de entrada a una respuesta de salida, sustituya el comodín # en el tema del mensaje con un ID de solicitud arbitrario. Por ejemplo, si publica un mensaje en `kinesisfirehose/message/binary/request123`, la propiedad `id` en el objeto de respuesta se establece en `request123`.

Si no desea asignar una solicitud a una respuesta, puede publicar sus mensajes en `kinesisfirehose/message/binary/`. Asegúrese de incluir la barra diagonal.

Version 1

Filtro de temas: `kinesisfirehose/message`

Consulte este tema para enviar un mensaje que contenga datos JSON.

Propiedades de mensajes

`request`

Los datos para enviar a la transmisión de entrega y a la transmisión de entrega de destino, si no es la transmisión predeterminada.

Obligatorio: `true`

Escriba: `object` que incluye las siguientes propiedades:

`data`

Los datos que enviar a la transmisión de entrega.

Obligatorio: `true`

Tipo: `string`

`delivery_stream_arn`

El ARN de la transmisión de entrega de Kinesis de destino. Incluya esta propiedad para anular la transmisión de entrega predeterminada.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a una respuesta de salida. Si se especifica, la propiedad `id` en el objeto de respuesta se establece en este valor. Si no utiliza esta característica, puede omitir esta propiedad o especificar una cadena vacía.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `.*`

Ejemplo de entrada

```
{
     "request": {
        "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
        "data": "Data to send to the delivery stream."
     },
     "id": "request123"
}
```

Filtro de temas: `kinesisfirehose/message/binary/#`

Consulte este tema para enviar un mensaje que contenga datos binarios. El conector no analiza los datos binarios. Los datos se transmiten tal cual.

Para asignar la solicitud de entrada a una respuesta de salida, sustituya el comodín # en el tema del mensaje con un ID de solicitud arbitrario. Por ejemplo, si publica un mensaje en `kinesisfirehose/message/binary/request123`, la propiedad `id` en el objeto de respuesta se establece en `request123`.

Si no desea asignar una solicitud a una respuesta, puede publicar sus mensajes en `kinesisfirehose/message/binary/`. Asegúrese de incluir la barra diagonal.

Datos de salida

Este conector publica información de estado como datos de salida en un tema MQTT.

Versions 2 - 5

Filtro de temas en la suscripción

```
kinesisfirehose/message/status
```

Ejemplo de salida

La respuesta contiene el estado de cada registro de datos enviado en el lote.

```
{
    "response": [
        {
            "ErrorCode": "error",
            "ErrorMessage": "test error",
            "id": "request123",
            "status": "fail"
        },
        {
            "firehose_record_id": "xyz2",
            "id": "request456",
            "status": "success"
        },
        {
            "firehose_record_id": "xyz3",
            "id": "request890",
            "status": "success"
        }
    ]
}
```

> **Note**
>
> Si el conector detecta un error que se puede volver a intentar (por ejemplo, errores de conexión), volverá a intentar la publicación en el siguiente lote. El SDK gestiona el retroceso exponencial. AWS Las solicitudes que fallan con errores que se pueden reintentar se añaden de nuevo al final de la cola para seguir publicándolos.

Version 1

Filtro de temas en la suscripción

```
kinesisfirehose/message/status
```

Ejemplo de salida: Correcto

```
{
   "response": {
       "firehose_record_id": "1lxfuuuFomkpJYzt/34ZU/r8JYPf8Wyf7AXqlXm",
       "status": "success"
    },
    "id": "request123"
}
```

Ejemplo de salida: Error

```
{
   "response" : {
       "error": "ResourceNotFoundException",
       "error_message": "An error occurred (ResourceNotFoundException) when
 calling the PutRecord operation: Firehose test1 not found under account
 123456789012.",
       "status": "fail"
   },
   "id": "request123"
}
```

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.

- Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.

   Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de AWS IoT Greengrass Core para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.
3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el conector y configure sus parámetros.

   c. Agregue suscripciones que permitan al conector recibir datos de entrada JSON y enviar datos de salida en filtros de tema compatibles.

      - Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
      - Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utiliza esta suscripción para ver los mensajes de estado en la AWS IoT consola.
4. Implemente el grupo.
5. En la AWS IoT consola, en la página de prueba, suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la característica envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector. Este mensaje contiene datos JSON.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'kinesisfirehose/message'

def create_request_with_all_fields():
    return  {
        "request": {
            "data": "Message from Firehose Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licencias

El conector de Kinesis Firehose incluye las siguientes licencias y software de terceros:

- AWS SDK for Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0

- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registros de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
5	Se ha agregado el parámetro `IsolationMode` para configurar el modo de creación de contenedores del conector.
4	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
3	Se ha corregido para reducir el registro excesivo y otras correcciones de errores menores.
2	Se agregó soporte para enviar registros de datos por lotes a Firehose en un intervalo específico. • Requiere además la acción `firehose:PutRecordBatch` en el rol de grupo. • Nuevos parámetros `MemorySize`, `DeliveryStreamQueueSize` y `PublishInterval`.

Versión	Cambios
	• El mensaje de salida contiene una matriz de respuestas de estado para los registros de datos publicados.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- ¿Qué es Amazon Kinesis Data Firehose? en la Guía para desarrolladores de Amazon Kinesis

Conector de ML Feedback

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

El conector ML Feedback facilita el acceso a los datos del modelo de machine learning (ML) para volver a entrenar y analizar modelos. El conector:

- Carga los datos de entrada (muestras) utilizados por el modelo de ML en Amazon S3. La entrada del modelo puede estar en cualquier formato, como imágenes, JSON o audio. Después de cargar muestras en la nube, puede utilizarlas para volver a entrenar el modelo y mejorar la precisión y exactitud de sus predicciones. Por ejemplo, puede utilizar SageMaker Ground Truth para etiquetar muestras y SageMaker para volver a entrenar el modelo.

- Publica los resultados de la predicción del modelo como mensajes MQTT. Esto le permite monitorizar y analizar la calidad de inferencia de su modelo en tiempo real. También puede almacenar los resultados de las predicciones y utilizarlos para analizar las tendencias a lo largo del tiempo.
- Publica métricas sobre cargas de muestras y datos de muestras en Amazon CloudWatch.

Para configurar este conector, describa las configuraciones de comentarios admitidas en formato JSON. Una configuración de retroalimentación define propiedades como el bucket de Amazon S3 de destino, el tipo de contenido y la estrategia de muestreo. Se utiliza una estrategia de muestreo para determinar qué muestras cargar.

Puede utilizar el conector ML Feedback en los siguientes escenarios:

- Con funciones de Lambda definidas por el usuario. Sus funciones de Lambda de inferencia local utilizan el SDK de machine learning AWS IoT Greengrass para invocar este conector y pasar la configuración de la retroalimentación de destino, la entrada del modelo y la salida del modelo (resultados de la predicción). Para ver un ejemplo, consulte the section called “Ejemplo de uso”.
- Con el conector de la clasificación de ML Image (v2) Para utilizar este conector con el conector de clasificación de ML Image, configure el parámetro `MLFeedbackConnectorConfigId` para el conector de clasificación de ML Image.
- Con el conector de detección de ML Object Para utilizar este conector con el conector de detección de ML Object, configure el parámetro `MLFeedbackConnectorConfigId` para el conector de detección de ML Object.

ARN: `arn:aws:greengrass:region::/connectors/MLFeedback/versions/1`

Requisitos

Este conector exige los siguientes requisitos:

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

> **Note**
>
> Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Uno o más buckets de Amazon S3. El número de buckets que utilice depende de su estrategia de muestreo.
- El rol del grupo de Greengrass configurado para permitir la acción `s3:PutObject` en los objetos del bucket de Amazon S3 de destino, tal como se muestra en la siguiente política de IAM de ejemplo.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

La política debe incluir todos los buckets de destino como recursos. Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *)

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

- El conector CloudWatch Metrics añadido al grupo de Greengrass y configurado. Esto solo es necesario si desea utilizar la característica de informes de métricas.
- Se requiere el SDK AWS IoT Greengrass de machine learning versión 1.1.0 para interactuar con este conector.

Parámetros

`FeedbackConfigurationMap`

Un conjunto de una o varias configuraciones de comentarios que el conector puede utilizar para cargar muestras en Amazon S3. Una configuración de comentarios define parámetros como el bucket de destino, el tipo de contenido y la estrategia de muestreo. Cuando se invoca este conector, el conector o la función de Lambda que realiza la llamada especifica una configuración de comentarios de destino.

Nombre para mostrar en la consola AWS IoT: Mapa de configuración de comentarios

Obligatorio: `true`

Tipo: una cadena JSON bien formada que define el conjunto de configuraciones de comentarios admitidas. Para ver un ejemplo, consulte the section called “Ejemplo de FeedbackConfigurationMap”.

El ID de un objeto de configuración de comentarios tiene los siguientes requisitos.

El ID:

- Debe ser único entre los objetos de configuración.
- Debe comenzar por una letra o un número. Puede contener letras minúsculas y mayúsculas, números y guiones.
- Debe tener entre 2 y 63 caracteres de longitud.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Ejemplos: `MyConfig0`, `config-a` y `12id`

El cuerpo de un objeto de configuración de comentarios contiene las siguientes propiedades.

`s3-bucket-name`

El nombre del bucket de Amazon S3 de destino.

> **Note**
>
> El rol de grupo debe permitir la acción `s3:PutObject` en todos los buckets de destino. Para obtener más información, consulte the section called “Requisitos”.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[a-z0-9\.\-]{3,63}$`

`content-type`

El tipo de contenido de las muestras que se van a cargar. Todo el contenido de una configuración de retroalimentación individual debe ser del mismo tipo.

Obligatorio: `true`

Escriba: `string`

Ejemplos: `image/jpeg`, `application/json` y `audio/ogg`

`s3-prefix`

El prefijo de clave que se va a utilizar para las muestras cargadas. Un prefijo es similar a un nombre de directorio. Le permite almacenar datos similares en el mismo directorio en un bucket. Para obtener más información, consulte Clave de objeto y metadatos en la Guía del usuario de Amazon Simple Storage Service.

Obligatorio: `false`

Escriba: `string`

`file-ext`

La extensión de archivo que se va a utilizar para las muestras cargadas. Debe ser una extensión de archivo válida para el tipo de contenido.

Obligatorio: `false`

Escriba: `string`

Ejemplos: `jpg`, `json` y `ogg`

`sampling-strategy`

La estrategia de muestreo que se va a utilizar para filtrar qué muestras cargar. Si se omite, el conector intenta cargar todas las muestras que recibe.

Obligatorio: `false`

Tipo: una cadena JSON bien formada que contiene las siguientes propiedades.

`strategy-name`

El nombre de la estrategia de muestreo.

Obligatorio: `true`

Escriba: `string`

Valores válidos: `RANDOM_SAMPLING`, `LEAST_CONFIDENCE`, `MARGIN` o `ENTROPY`

`rate`

La velocidad de la estrategia de muestreo Random (Aleatorio).

Obligatorio: `true` si `strategy-name` es `RANDOM_SAMPLING`.

Escriba: `number`

Valores válidos: `0.0 - 1.0`

`threshold`

El umbral de la estrategia de muestreo Least Confidence (Confianza mínima), Margin (Margen) o Entropy (Entropía).

Obligatorio: `true` si `strategy-name` es `LEAST_CONFIDENCE`, `MARGIN` o `ENTROPY`.

Escriba: `number`

Valores válidos:

- `0.0 - 1.0` para la estrategia `LEAST_CONFIDENCE` o `MARGIN`.
- `0.0 - no limit` para la estrategia `ENTROPY`.

`RequestLimit`

El número máximo de solicitudes que el conector puede procesar a la vez.

Puede utilizar este parámetro para restringir el consumo de memoria limitando el número de solicitudes que el conector procesa al mismo tiempo. Se ignorarán las solicitudes que superen ese límite.

Nombre para mostrar en la consola AWS IoT: Límite de solicitudes

Obligatorio: `false`

Escriba: `string`

Valores válidos: `0 - 999`

Patrón válido: `^$|^[0-9]{1,3}$`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector de ML Feedback.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyMLFeedbackConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/MLFeedback/
versions/1",
            "Parameters": {
                "FeedbackConfigurationMap": "{  \"RandomSamplingConfiguration\":
 {  \"s3-bucket-name\": \"my-aws-bucket-random-sampling\",  \"content-type\":
 \"image/png\",  \"file-ext\": \"png\",  \"sampling-strategy\": {  \"strategy-name
\": \"RANDOM_SAMPLING\",  \"rate\": 0.5  } },  \"LeastConfidenceConfiguration\": {
  \"s3-bucket-name\": \"my-aws-bucket-least-confidence-sampling\",  \"content-type\":
 \"image/png\",  \"file-ext\": \"png\",  \"sampling-strategy\": {  \"strategy-name\":
 \"LEAST_CONFIDENCE\",  \"threshold\": 0.4  } } }",
                "RequestLimit": "10"
```

```
            }
        }
    ]
}'
```

Ejemplo de FeedbackConfigurationMap

A continuación se muestra un valor de ejemplo ampliado para el parámetro `FeedbackConfigurationMap`. Este ejemplo incluye varias configuraciones de comentarios que utilizan diferentes estrategias de muestreo.

```
{
    "ConfigID1": {
        "s3-bucket-name": "my-aws-bucket-random-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "RANDOM_SAMPLING",
            "rate": 0.5
        }
    },
    "ConfigID2": {
        "s3-bucket-name": "my-aws-bucket-margin-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "MARGIN",
            "threshold": 0.4
        }
    },
    "ConfigID3": {
        "s3-bucket-name": "my-aws-bucket-least-confidence-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "LEAST_CONFIDENCE",
            "threshold": 0.4
        }
    },
    "ConfigID4": {
        "s3-bucket-name": "my-aws-bucket-entropy-sampling",
        "content-type": "image/png",
        "file-ext": "png",
```

```
        "sampling-strategy": {
            "strategy-name": "ENTROPY",
            "threshold": 2
        }
    },
    "ConfigID5": {
        "s3-bucket-name": "my-aws-bucket-no-sampling",
        "s3-prefix": "DeviceA",
        "content-type": "application/json"
    }
}
```

Estrategias de muestreo

El conector admite cuatro estrategias de muestreo que determinan si cargar muestras que se transfieren al conector. Las muestras son instancias discretas de datos que un modelo utiliza para una predicción. Puede utilizar estrategias de muestreo para filtrar por las muestras que es más probable que mejoren la precisión del modelo.

`RANDOM_SAMPLING`

Carga aleatoriamente muestras en función de la velocidad proporcionada. Carga una muestra si un valor generado aleatoriamente es inferior a la velocidad. Cuanto mayor sea la velocidad, más muestras se cargarán.

> **Note**
>
> Esta estrategia hace caso omiso de cualquier predicción de modelo que se proporcione.

`LEAST_CONFIDENCE`

Carga muestras cuya probabilidad de confianza máxima sea inferior al umbral proporcionado.

Ejemplo de escenarios:

Umbral: `.6`

Predicción del modelo: `[.2, .2, .4, .2]`

Probabilidad de confianza máxima: `.4`

Resultado:

Utilice el muestreo, ya que la probabilidad de confianza máxima (`.4`) es inferior o igual al umbral (`.6`).

`MARGIN`

Carga muestras si el margen entre las dos primeras probabilidades de confianza se encuentra dentro del umbral proporcionado. El margen es la diferencia entre las dos primeras probabilidades.

Ejemplo de escenarios:

Umbral: `.02`

Predicción del modelo: `[.3, .35, .34, .01]`

Dos probabilidades de confianza principales: `[.35, .34]`

Margen: `.01` (`.35 - .34`)

Resultado:

Utilice la muestra, ya que el margen (`.01`) es inferior o igual que el umbral (`.02`).

`ENTROPY`

Carga muestras cuya entropía es mayor que el umbral proporcionado. Utiliza la entropía normalizada de la predicción del modelo.

Ejemplo de escenarios:

Umbral: `0.75`

Predicción del modelo: `[.5, .25, .25]`

Entropía para la predicción: `1.03972`

Resultado:

Utilice la muestra porque la entropía (`1.03972`) es superior al umbral (`0.75`).

Datos de entrada

Las funciones de Lambda definidas por el usuario utilizan la función `publish` del cliente `feedback` en el SDK AWS IoT Greengrass de machine learning para invocar al conector. Para ver un ejemplo, consulte the section called “Ejemplo de uso”.

> **Note**
>
> Este conector no acepta mensajes MQTT como datos de entrada.

La función `publish` acepta los argumentos siguientes:

ConfigId

El ID de la configuración de comentarios de destino. Debe coincidir con el ID de una configuración de realimentación definida en el parámetro FeedbackConfigurationMap para el conector de ML Feedback.

Obligatorio: true

Tipo: cadena

ModelInput

Los datos de entrada que se pasaron a un modelo para la inferencia. Estos datos de entrada se cargan utilizando la configuración de destino a menos que se filtren en función de la estrategia de muestreo.

Obligatorio: true

Tipo: bytes

ModelPrediction

La predicción se genera a partir del modelo. El tipo de resultado puede ser un diccionario o una lista. Por ejemplo, los resultados de predicción del conector de clasificación de ML Image es una lista de probabilidades (como `[0.25, 0.60, 0.15]`). Estos datos se publican en el tema `/feedback/message/prediction`.

Obligatorio: true

Tipo: diccionario o lista de valores `float`

Metadatos

Metadatos específicos de la aplicación definidos por el cliente que se añaden a la muestra cargada y se publican en el tema `/feedback/message/prediction`. El conector también inserta una clave `publish-ts` con un valor de marca temporal en los metadatos.

Obligatorio: false

Tipo: diccionario

Ejemplo: `{"some-key": "some value"}`

Datos de salida

Este conector publica datos en tres temas MQTT:

- La información del estado del conector en el tema `feedback/message/status`.
- Resultados de predicción en el tema `feedback/message/prediction`.
- Métricas destinadas a CloudWatch en el tema `cloudwatch/metric/put`.

Debe configurar las suscripciones para permitir la comunicación del conector en temas de MQTT. Para obtener más información, consulte the section called “Entradas y salidas”.

Filtro de temas: `feedback/message/status`

Utilice este tema para monitorizar el estado de las cargas de muestra y las muestras eliminadas. El conector publica en este tema cada vez que recibe una solicitud.

Ejemplo del resultado: carga de muestra realizada correctamente


```
{
  "response": {
    "status": "success",
    "s3_response": {
      "ResponseMetadata": {
        "HostId": "IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK
+Jd1vEXAMPLEa3Km",
        "RetryAttempts": 1,
        "HTTPStatusCode": 200,
        "RequestId": "79104EXAMPLEB723",
        "HTTPHeaders": {
          "content-length": "0",
          "x-amz-id-2":
 "lbbqaDVFOhMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
          "server": "AmazonS3",
          "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
 rule-id=\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
          "x-amz-request-id": "79104EXAMPLEB723",
```

```
          "etag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
          "date": "Thu, 11 Jul 2019 00:12:50 GMT",
          "x-amz-server-side-encryption": "AES256"
        }
      },
      "bucket": "greengrass-feedback-connector-data-us-west-2",
      "ETag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
      "Expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\", rule-id=
\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
      "key": "s3-key-prefix/UUID.file_ext",
      "ServerSideEncryption": "AES256"
    }
  },
  "id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
}
```

El conector añade los campos `bucket` y `key` a la respuesta desde Amazon S3. Para obtener más información sobre la respuesta de Amazon S3, consulte Objeto PUT en la Referencia de la API de Amazon Simple Storage Service..

Ejemplo del resultado: muestra eliminada debido a la estrategia de muestreo

```
{
  "response": {
    "status": "sample_dropped_by_strategy"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Ejemplo de resultado: error al cargar el ejemplo

Un estado de error incluye el mensaje de error como el valor `error_message` y la clase de excepción como el valor `error`.

```
{
  "response": {
    "status": "fail",
    "error_message": "[RequestId: 4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3] Failed
 to upload model input data due to exception. Model prediction will not be
 published. Exception type: NoSuchBucket, error: An error occurred (NoSuchBucket)
 when calling the PutObject operation: The specified bucket does not exist",
    "error": "NoSuchBucket"
  },
```

```
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Ejemplo del resultado: solicitud limitada debido al límite de solicitudes

```
{
  "response": {
    "status": "fail",
    "error_message": "Request limit has been reached (max request: 10 ). Dropping
 request.",
    "error": "Queue.Full"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Filtro de temas: `feedback/message/prediction`

Utilice este tema para escuchar predicciones basadas en datos de muestra cargados. Esto le permite analizar el rendimiento del modelo en tiempo real. Las predicciones de modelos se publican en este tema solo si los datos se cargan correctamente en Amazon S3. Los mensajes publicados en este tema están en formato JSON. Contienen el enlace al objeto de datos cargado, la predicción del modelo y los metadatos incluidos en la solicitud.

También puede almacenar los resultados de las predicciones y utilizarlos para analizar las tendencias a lo largo del tiempo e informar de ellas. Las tendencias pueden proporcionar información valiosa. Por ejemplo, una reducción de la precisión a lo largo del tiempo puede ayudarle a decidir si el modelo debe volver a entrenarse.

Ejemplo de salida

```
{
  "source-ref": "s3://greengrass-feedback-connector-data-us-west-2/s3-key-prefix/
UUID.file_ext",
  "model-prediction": [
    0.5,
    0.2,
    0.2,
    0.1
  ],
  "config-id": "ConfigID2",
  "metadata": {
    "publish-ts": "2019-07-11 00:12:48.816752"
```

```
  }
}
```

> **Tip**
>
> Puede configurar el conector de IoT Analytics para suscribirse a este tema y enviar información a AWS IoT Analytics para obtener un análisis adicional o histórico.

Filtro de temas: `cloudwatch/metric/put`

Este es el tema de salida que se utiliza para publicar métricas en CloudWatch. Esta característica requiere que instale y configure el conector CloudWatch Metrics.

Entre las métricas se incluyen:

- El número de muestras cargadas.
- El tamaño de las muestras cargadas.
- El número de errores de cargas en Amazon S3.
- El número de muestras eliminadas en función de la estrategia de muestreo.
- El número de solicitudes restringidas.

Ejemplo de salida: tamaño de la muestra de datos (publicada antes de la carga real)

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 47592,
      "unit": "Bytes",
      "metricName": "SampleSize"
    }
  }
}
```

Ejemplo del resultado: carga de muestra realizada correctamente

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
```

```
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadSuccess"
    }
  }
}
```

Ejemplo de resultado: carga de muestra realizada correctamente y resultado de predicción publicado

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleAndPredictionPublished"
    }
  }
}
```

Ejemplo de resultado: error al cargar el ejemplo

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadFailure"
    }
  }
}
```

Ejemplo del resultado: muestra eliminada debido a la estrategia de muestreo

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
```

```
      "metricName": "SampleNotUsed"
    }
  }
}
```

Ejemplo del resultado: solicitud limitada debido al límite de solicitudes

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "ErrorRequestThrottled"
    }
  }
}
```

Ejemplo de uso

El siguiente ejemplo es una función de Lambda definida por el usuario que utiliza el SDK AWS IoT Greengrass de machine learning para enviar datos al conector ML Feedback.

> **Note**
>
> Puede descargar el SDK de AWS IoT Greengrass Machine Learning desde la página de AWS IoT Greengrass descargas.

```
import json
import logging
import os
import sys
import greengrass_machine_learning_sdk as ml

client = ml.client('feedback')

try:
    feedback_config_id = os.environ["FEEDBACK_CONFIG_ID"]
    model_input_data_dir = os.environ["MODEL_INPUT_DIR"]
    model_prediction_str = os.environ["MODEL_PREDICTIONS"]
```

```
    model_prediction = json.loads(model_prediction_str)
except Exception as e:
    logging.info("Failed to open environment variables. Failed with exception:
{}".format(e))
    sys.exit(1)

try:
    with open(os.path.join(model_input_data_dir, os.listdir(model_input_data_dir)[0]),
 'rb') as f:
        content = f.read()
except Exception as e:
    logging.info("Failed to open model input directory. Failed with exception:
{}".format(e))
    sys.exit(1)

def invoke_feedback_connector():
    logging.info("Invoking feedback connector.")
    try:
        client.publish(
            ConfigId=feedback_config_id,
            ModelInput=content,
            ModelPrediction=model_prediction
        )
    except Exception as e:
        logging.info("Exception raised when invoking feedback connector:{}".format(e))
        sys.exit(1)

invoke_feedback_connector()

def function_handler(event, context):
    return
```

Licencias

El conector de ML Feedback incluye las siguientes licencias y software de terceros:

- AWS SDK for Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain

- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

- six/MIT

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”

Conector de la clasificación de ML Image

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

Los conectores Image Classification de ML proporcionan un servicio de inferencia de machine learning (ML) que se ejecuta en el núcleo de AWS IoT Greengrass. Este servicio de inferencia local realiza clasificación de imágenes mediante un modelo entrenado por el algoritmo de clasificación de imágenes de .SageMaker.

Las funciones de Lambda definidas por el usuario utilizan el SDK de AWS IoT Greengrass de machine learning para enviar solicitudes de inferencia al servicio de inferencia local. El servicio ejecuta la inferencia localmente y devuelve las probabilidades de que la imagen de entrada pertenezca a categorías específicas.

AWS IoT Greengrass proporciona las siguientes versiones de este conector, que está disponible para varias plataformas.

Version 2

Kinesis - S3	Descripción y ARN
Clasificación de imágenes de ML Aarch64 JTX2	Servicio de inferencia para clasificación de imágenes para NVIDIA Jetson TX2. Admite aceleración de GPU. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationAarch64JTX2/versions/2`
Clasificación de imágenes de ML x86_64	Servicio de inferencia para clasificación de imágenes para plataformas x86_64. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationx86-64/versions/2`
Clasificación de imágenes de ML ARMv7	Servicio de inferencia para clasificación de imágenes para plataformas ARMv7. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationARMv7/versions/2`

Version 1

Kinesis - S3	Descripción y ARN
Clasificación de imágenes de ML Aarch64 JTX2	Servicio de inferencia para clasificación de imágenes para NVIDIA Jetson TX2. Admite aceleración de GPU.

Kinesis - S3	Descripción y ARN
	ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationAarch64JTX2/versions/1`
Clasificación de imágenes de ML x86_64	Servicio de inferencia para clasificación de imágenes para plataformas x86_64. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationx86-64/versions/1`
Clasificación de imágenes de ML Armv7	Servicio de inferencia para clasificación de imágenes para plataformas Armv7. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationARMv7/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Estos conectores tienen los siguientes requisitos:

Version 2

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

> **Note**
>
> Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Dependencias para el marco de trabajo Apache MXNet instalado en el dispositivo de núcleo. Para obtener más información, consulte the section called “Instalación de dependencias de MXNet”.
- Un recurso ML en el grupo de Greengrass que hace referencia a un origen del modelo SageMaker. Este modelo debe ser entrenado por el algoritmo de clasificación de imágenes de SageMaker. Para obtener más información, consulte Algoritmo de clasificación de imágenes en la Guía para desarrolladores de Amazon SageMaker.
- El conector de Feedback de ML añadido al grupo de Greengrass y configurado. Solo es necesario si desea utilizar el conector para cargar datos de entrada del modelo y publicar predicciones en un tema de MQTT.
- El rol del grupo de Greengrass configurado para permitir la acción `sagemaker:DescribeTrainingJob` en la tarea de entrenamiento de destino, tal y como se muestra en la siguiente política de IAM de ejemplo.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Resource": "arn:aws:sagemaker:region:account-id:training-
job:training-job-name"
        }
    ]
```

```
}
```

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Si cambia el trabajo de entrenamiento de destino en el futuro, asegúrese de actualizar el rol de grupo.

- Se requiere el SDK AWS IoT Greengrass de machine learning versión 1.1.0 para interactuar con este conector.

Version 1

- Software AWS IoT Greengrass Core versión 1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Dependencias para el marco de trabajo Apache MXNet instalado en el dispositivo de núcleo. Para obtener más información, consulte the section called “Instalación de dependencias de MXNet”.
- Un recurso ML en el grupo de Greengrass que hace referencia a un origen del modelo SageMaker. Este modelo debe ser entrenado por el algoritmo de clasificación de imágenes de SageMaker. Para obtener más información, consulte Algoritmo de clasificación de imágenes en la Guía para desarrolladores de Amazon SageMaker.
- El rol del grupo de Greengrass configurado para permitir la acción `sagemaker:DescribeTrainingJob` en la tarea de entrenamiento de destino, tal y como se muestra en la siguiente política de IAM de ejemplo.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
```

```
            "Resource": "arn:aws:sagemaker:region:account-id:training-
job:training-job-name"
        }
    ]
}
```

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Si cambia el trabajo de entrenamiento de destino en el futuro, asegúrese de actualizar el rol de grupo.

- Se requiere el SDK de machine learning deAWS IoT Greengrass versión 1.1.0 para interactuar con este conector.

Parámetros de conector

Estos conectores proporcionan los siguientes parámetros.

Version 2

`MLModelDestinationPath`

Es la ruta local absoluta del recurso ML dentro del entorno de Lambda. Esta es la ruta de destino que se ha especificado para el recurso de aprendizaje automático.

> **Note**
>
> Si ha creado el recurso de aprendizaje automático en la consola, esta es la ruta local.

Nombre que mostrar en la consola AWS IoT: Ruta de destino del modelo

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`MLModelResourceId`

El ID del recurso de aprendizaje automático que hace referencia al modelo de origen.

Nombre para mostrar en la consola AWS IoT: SageMaker job ARN resource

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

El ARN del trabajo de formación de SageMaker que representa el origen del modelo de SageMaker. El modelo debe ser entrenado por el algoritmo de clasificación de imágenes de SageMaker.

Nombre para mostrar en la consola AWS IoT: Recurso ARN del trabajo de SageMaker

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

El nombre del servicio de inferencia local. Las funciones de Lambda definidas por el usuario invocan el servicio pasando el nombre a la función `invoke_inference_service` del SDK de machine learning de AWS IoT Greengrass. Para ver un ejemplo, consulte the section called “Ejemplo de uso”.

Nombre que mostrar en la consola AWS IoT: Nombre del servicio de inferencia local

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

La cantidad de tiempo (en segundos) antes de que se termine la solicitud de inferencia. El valor mínimo es 1.

Nombre que mostrar en la consola AWS IoT: Tiempo de espera (segundo)

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

La cantidad de memoria (en KB) a la que el servicio tiene acceso. El valor mínimo es 1.

Nombre que mostrar en la consola AWS IoT: Límite de memoria (KB)

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[1-9][0-9]*`

`GPUAcceleration`

El contexto de informática de la CPU o GPU (acelerada). Esta propiedad se aplica al solo al conector Image Classification de ML Aarch64 JTX2.

Nombre que mostrar en la consola AWS IoT: Aceleración de la GPU

Obligatorio: `true`

Escriba: `string`

Valores válidos: `CPU` o `GPU`

`MLFeedbackConnectorConfigId`

El ID de la configuración de retroalimentación que se va a utilizar para cargar los datos de entrada del modelo. Debe coincidir con el ID de una configuración de comentarios definida para el conector de comentarios de aprendizaje automático.

Este parámetro solo es necesario si desea utilizar el conector de comentarios de aprendizaje automático para cargar datos de entrada del modelo y publicar predicciones en un tema de MQTT.

Nombre que mostrar en la consola AWS IoT: ID de configuración del conector de ML Feedback

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Version 1

`MLModelDestinationPath`

Es la ruta local absoluta del recurso ML dentro del entorno de Lambda. Esta es la ruta de destino que se ha especificado para el recurso de aprendizaje automático.

> **Note**
>
> Si ha creado el recurso de aprendizaje automático en la consola, esta es la ruta local.

Nombre que mostrar en la consola AWS IoT: Ruta de destino del modelo

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`MLModelResourceId`

El ID del recurso de aprendizaje automático que hace referencia al modelo de origen.

Nombre para mostrar en la consola AWS IoT: SageMaker job ARN resource

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

El ARN del trabajo de formación de SageMaker que representa el origen del modelo de SageMaker. El modelo debe ser entrenado por el algoritmo de clasificación de imágenes de SageMaker.

Nombre para mostrar en la consola AWS IoT: Recurso ARN del trabajo de SageMaker

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

El nombre del servicio de inferencia local. Las funciones de Lambda definidas por el usuario invocan el servicio pasando el nombre a la función `invoke_inference_service` del SDK de machine learning de AWS IoT Greengrass. Para ver un ejemplo, consulte the section called "Ejemplo de uso".

Nombre que mostrar en la consola AWS IoT: Nombre del servicio de inferencia local

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

La cantidad de tiempo (en segundos) antes de que se termine la solicitud de inferencia. El valor mínimo es 1.

Nombre que mostrar en la consola AWS IoT: Tiempo de espera (segundo)

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

La cantidad de memoria (en KB) a la que el servicio tiene acceso. El valor mínimo es 1.

Nombre que mostrar en la consola AWS IoT: Límite de memoria (KB)

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[1-9][0-9]*`

`GPUAcceleration`

El contexto de informática de la CPU o GPU (acelerada). Esta propiedad se aplica al solo al conector Image Classification de ML Aarch64 JTX2.

Nombre que mostrar en la consola AWS IoT: Aceleración de la GPU

Obligatorio: `true`

Escriba: `string`

Valores válidos: `CPU` o `GPU`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea un parámetro `ConnectorDefinition` con una versión inicial que contiene un conector Image Classification de ML.

Ejemplo: instancia de CPU

Este ejemplo crea una instancia del conector Image Classification de ML Armv7l.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationARMv7/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'
```

Ejemplo: instancia de GPU

En este ejemplo se crea una instancia del conector Image Classification de ML Aarch64 JTX2, que admite la aceleración de GPU en una tarjeta NVIDIA Jetson TX2.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationAarch64JTX2/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "GPUAcceleration": "GPU",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de estos conectores tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de Conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Estos conectores aceptan un archivo de imagen como entrada. Los archivos de imagen de entrada deben tener el formato `jpeg` o `png`. Para obtener más información, consulte the section called “Ejemplo de uso”.

Estos conectores no aceptan mensajes MQTT como datos de entrada.

Datos de salida

Estos conectores devuelven una predicción con formato para el objeto identificado en la imagen de entrada:

```
[0.3,0.1,0.04,...]
```

La predicción contiene una lista de valores que se corresponden con las categorías utilizadas en el conjunto de datos de entrenamiento durante la capacitación de modelos. Cada valor representa la probabilidad de que la imagen se encuentre dentro de la categoría correspondiente. La categoría con la mayor probabilidad es la predicción dominante.

Estos conectores no publican mensajes MQTT como datos de salida.

Ejemplo de uso

El siguiente ejemplo de función de Lambda utiliza el SDK de Machine Learning AWS IoT Greengrass para interactuar con un conector de Image Classification de ML.

> **Note**
>
> Puede descargar el SDK desde la página de descargas del SDK de machine learning de AWS IoT Greengrass.

En el ejemplo se inicializa un cliente SDK y de forma sincrónica llama a la función `invoke_inference_service` del SDK para invocar el servicio de inferencia local. Pasa el tipo de algoritmo, el nombre del servicio, el tipo de imagen y contenido de imágenes. A continuación, el ejemplo analiza la respuesta del servicio para obtener los resultados de probabilidad (predicciones).

Python 3.7

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml
```

```
# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='image-classification',
            ServiceName='imageClassification',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}("{}")'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}("{}")'.format(e.__class__.__name__,
 e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    count = len(predictions.split(','))
    predictions_arr = np.fromstring(predictions, count=count, sep=',')

    # Perform business logic that relies on the predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()
    return

infer()
```

```
def function_handler(event, context):
    return
```

Python 2.7

```
import logging
from threading import Timer

import numpy

import greengrass_machine_learning_sdk as gg_ml

# The inference input image.
with open("/test_img/test.jpg", "rb") as f:
    content = f.read()

client = gg_ml.client("inference")


def infer():
    logging.info("Invoking Greengrass ML Inference service")

    try:
        resp = client.invoke_inference_service(
            AlgoType="image-classification",
            ServiceName="imageClassification",
            ContentType="image/jpeg",
            Body=content,
        )
    except gg_ml.GreengrassInferenceException as e:
        logging.info('Inference exception %s("%s")', e.__class__.__name__, e)
        return
    except gg_ml.GreengrassDependencyException as e:
        logging.info('Dependency exception %s("%s")', e.__class__.__name__, e)
        return

    logging.info("Response: %s", resp)
    predictions = resp["Body"].read()
    logging.info("Predictions: %s", predictions)

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
```

```
    predictions_arr = numpy.fromstring(predictions, sep=",")
    logging.info("Split into %s predictions.", len(predictions_arr))

    # Perform business logic that relies on predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()


infer()


# In this example, the required AWS Lambda handler is never called.
def function_handler(event, context):
    return
```

La función `invoke_inference_service` en el SDK de machine learning de AWS IoT Greengrass acepta los argumentos siguientes.

Argumento	Descripción
`AlgoType`	El nombre del tipo de algoritmo que usar para la interferencia. En la actualidad, solo se admite `image-classification` . Obligatorio: `true` Escriba: `string` Valores válidos: `image-classification`
`ServiceName`	El nombre del servicio de inferencia local. Utilice el nombre que especificó para el parámetro `LocalInferenceServiceName` al configurar el conector. Obligatorio: `true`

Argumento	Descripción
	Escriba: `string`
`ContentType`	El tipo mime de la imagen de entrada. Obligatorio: `true` Escriba: `string` Valores válidos: `image/jpeg, image/png`
`Body`	El contenido del archivo de la imagen de entrada. Obligatorio: `true` Escriba: `binary`

Instalación de dependencias de MXNet en el núcleo de AWS IoT Greengrass

Para usar un conector Image Classification de ML debe instalar las dependencias para el marco de trabajo Apache MXNet del dispositivo de núcleo. Los conectores utilizan el marco de trabajo para servir al modelo de ML.

> **Note**
>
> Estos conectores están agrupados con una biblioteca MXNet precompilada, por lo que no tiene que instalar el marco de trabajo MXNet en sí en el dispositivo de núcleo.

AWS IoT Greengrass proporciona scripts para instalar las dependencias de las siguientes plataformas y dispositivos comunes (o para utilizarlos como referencia para instalarlos). Si utiliza una plataforma diferente o dispositivo diferentes, consulte la documentación de MXNet para su configuración.

Antes de instalar las dependencias de MXNet, asegúrese de que las bibliotecas del sistema requeridas (con la versiones mínimas especificadas) están presentes en el dispositivo.

NVIDIA Jetson TX2

1. Instalación del conjunto de herramientas CUDA 9.0 y cuDNN 7.0. Puede seguir las instrucciones en the section called “Configuración de otros dispositivos” en el tutorial de Introducción.
2. Habilitar repositorios universales para que el conector pueda instalar software abierto mantenido por la comunidad. Para obtener más información, consulte Repositories/Ubuntu en la documentación de Ubuntu.

   a. Abra el archivo `/etc/apt/sources.list`.

   b. Asegúrese de que las siguientes líneas no tienen comentarios.

   ```
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   ```

3. Guarde una copia del siguiente script de instalación en un archivo llamado `nvidiajtx2.sh` en el dispositivo del núcleo.

   Python 3.7

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."
   echo 'Assuming that universe repos are enabled and checking dependencies...'
   apt-get -y update
   apt-get -y dist-upgrade
   apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-
   dev
   apt-get install -y python3.7 python3.7-dev

   python3.7 -m pip install --upgrade pip
   python3.7 -m pip install numpy==1.15.0
   python3.7 -m pip install opencv-python || echo 'Error: Unable to install
    OpenCV with pip on this platform. Try building the latest OpenCV from source
    (https://github.com/opencv/opencv).'

   echo 'Dependency installation/upgrade complete.'
   ```

> **Note**
>
> Si OpenCV no se instala correctamente con este script, puede intentar realizar la compilación desde el código fuente. Para obtener más información, consulte Instalación en Linux en la documentación de OpenCV o vea otros recursos online para su plataforma.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
 python-dev

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install numpy==1.15.0 scipy

echo 'Dependency installation/upgrade complete.'
```

4. En el directorio en el que guardó el archivo, ejecute el siguiente comando:

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Guarde una copia del siguiente script de instalación en un archivo llamado `x86_64.sh` en el dispositivo del núcleo.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
  apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
  echo "OS Release not supported: $release"
  exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
 OpenCV with pip on this platform. Try building the latest OpenCV from source
 (https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> Si OpenCV no se instala correctamente con este script, puede intentar realizar la compilación desde el código fuente. Para obtener más información, consulte

Instalación en Linux en la documentación de OpenCV o vea otros recursos online para su plataforma.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1 python-dev
 python-pip
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext python-
pip
else
  echo "OS Release not supported: $release"
  exit 1
fi

pip install numpy==1.15.0 scipy opencv-python

echo 'Dependency installation/upgrade complete.'
```

2. En el directorio en el que guardó el archivo, ejecute el siguiente comando:

```
sudo x86_64.sh
```

Armv7 (Raspberry Pi)

1. Guarde una copia del siguiente script de instalación en un archivo llamado `armv7l.sh` en el dispositivo del núcleo.

   Python 3.7

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."

   apt-get update
   apt-get -y upgrade

   apt-get install -y liblapack3 libopenblas-dev liblapack-dev
   apt-get install -y python3.7 python3.7-dev

   python3.7 -m pip install --upgrade pip
   python3.7 -m pip install numpy==1.15.0
   python3.7 -m pip install opencv-python || echo 'Error: Unable to install
    OpenCV with pip on this platform. Try building the latest OpenCV from source
    (https://github.com/opencv/opencv).'

   echo 'Dependency installation/upgrade complete.'
   ```

   > **Note**
   >
   > Si OpenCV no se instala correctamente con este script, puede intentar realizar la compilación desde el código fuente. Para obtener más información, consulte Instalación en Linux en la documentación de OpenCV o vea otros recursos online para su plataforma.

   Python 2.7

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."
   ```

```
apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev python-dev

# python-opencv depends on python-numpy. The latest version in the APT
 repository is python-numpy-1.8.2
# This script installs python-numpy first so that python-opencv can be
 installed, and then install the latest
# numpy-1.15.x with pip
apt-get install -y python-numpy python-opencv
dpkg --remove --force-depends python-numpy

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install --upgrade numpy==1.15.0 picamera scipy

echo 'Dependency installation/upgrade complete.'
```

2. En el directorio en el que guardó el archivo, ejecute el siguiente comando:

```
sudo bash armv7l.sh
```

> **Note**
>
> En una solución Raspberry Pi, el uso de `pip` para instalar dependencias de aprendizaje automático es una operación de uso intensivo de memoria que puede provocar que el dispositivo se quede sin memoria y deje de responder. Como alternativa, puede aumentar temporalmente el tamaño de intercambio: En `/etc/dphys-swapfile`, aumente el valor de la variable `CONF_SWAPSIZE` y, a continuación, ejecute el siguiente comando para reiniciar `dphys-swapfile`.
>
> ```
> /etc/init.d/dphys-swapfile restart
> ```

Registro y solución de problemas

En función de su configuración del grupo, los registros de evento y error se escriben en el registro de CloudWatch, el sistema de archivos local o ambos. Los registros de este conector utilizan el prefijo `LocalInferenceServiceName`. Si el conector se comporta de forma inesperada, compruebe los registros del conector. Estos suelen contener información de depuración útil, como, por ejemplo, que falta una dependencia de biblioteca de ML o la causa de un error de inicio del conector.

Si el grupo AWS IoT Greengrass está configurado para escribir registros locales, el conector escribe los archivos de registro en *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/aws/`. Para obtener más información sobre los registros de Greengrass, consulte the section called “Monitorización con registros de AWS IoT Greengrass”.

Utilice la siguiente información como ayuda para solucionar problemas con los conectores Image Classification de ML.

Bibliotecas del sistema obligatorias

Las siguientes pestañas muestran las bibliotecas del sistema necesarias para cada conector Image Classification de ML.

ML Image Classification Aarch64 JTX2

Library	Versión mínima
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	no aplicable
libcudart.so.9.0	no aplicable
libcudnn.so.7	no aplicable
libcufft.so.9.0	no aplicable
libcurand.so.9.0	no aplicable
libcusolver.so.9.0	no aplicable

Library	Versión mínima
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Image Classification x86_64

Library	Versión mínima
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Image Classification Armv7

Library	Versión mínima
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7

Library	Versión mínima
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problemas

Síntoma	Solución
En una Raspberry Pi, se registra el siguiente mensaje de error y no está utilizando la cámara: `Failed to initialize libdc1394`	Ejecute el comando siguiente para deshabilitar el controlador: `sudo ln /dev/null /dev/raw1394` Esta operación es efímera y el enlace simbólico desaparecerá después del reinicio. Consulte el manual de su distribución de SO para obtener información acerca de cómo crear automátic amente el enlace al reiniciar.

Licencias

Los conectores Image Classification de ML incluyen las siguientes licencias y software de terceros:

- AWS SDK for Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License

- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

- Biblioteca de red neuronal profunda (DNNL)/Licencia Apache 2.0
- Biblioteca de tiempo de ejecución OpenMP*/Consulte Licencias de bibliotecas de tiempo de ejecución Intel OpenMP.
- mxnet/Licencia Apache 2.0
- six/MIT

Licencias de biblioteca de tiempo de ejecución Intel OpenMP. El tiempo de ejecución de Intel® OpenMP* dispone de licencia doble, con una licencia comercial (COM) como parte de los productos Intel® Parallel Studio XE Suite y una licencia de código abierto BSD (OSS).

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
2	Se ha añadido el parámetro `MLFeedbackConnectorConfigId` para admitir el uso del conector de Feedback de ML para cargar datos de entrada del modelo, publicar predicciones en un tema de MQTT y publicar métricas en Amazon CloudWatch.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- Cómo realizar la inferencia de machine learning
- Algoritmo de clasificación de imágenes en la Guía para desarrolladores de Amazon SageMaker

Conector de detección de ML Object

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

Los conectores Object Detection de ML proporcionan un servicio de inferencia de machine learning (ML) que se ejecuta en el núcleo de AWS IoT Greengrass. Este servicio de inferencia local realiza el recopilado por el compilador de aprendizaje profundo SageMaker Neo. Se admiten dos tipos de modelos de detección de objetos: Single Shot Multibox Detector (SSD) y You Only Look Once (YOLO) v3. Para obtener más información, consulte Requisitos del modelo de detección de objetos.

Las funciones de Lambda definidas por el usuario utilizan el SDK de AWS IoT Greengrass de machine learning para enviar solicitudes de inferencia al servicio de inferencia local. El servicio realiza la inferencia local en una imagen de entrada y devuelve una lista de predicciones para cada objeto detectado en la imagen. Cada predicción contiene una categoría de objeto, una puntuación de confianza de predicción y coordenadas de píxeles que especifican un cuadro delimitador alrededor del objeto previsto.

AWS IoT Greengrass proporciona conectores de detección de objetos de ML para múltiples plataformas:

Kinesis - S3	Descripción y ARN
Detección de objetos de ML Aarch64 JTX2	Servicio de inferencia de detección de objetos para NVIDIA Jetson TX2. Admite aceleración de GPU. ARN: `arn:aws:greengrass:region::/connectors/ObjectDetectionAarch64JTX2/versions/1`
Detección de objetos de ML x86_64	Servicio de inferencia de detección de objetos para plataformas x86_64. ARN: `arn:aws:greengrass:region::/connectors/ObjectDetectionx86-64/versions/1`
Detección de objetos de ML ARMv7	Servicio de inferencia de detección de objetos para plataformas ARMv7. ARN: `arn:aws:greengrass:region::/connectors/ObjectDetectionARMv7/versions/1`

Requisitos

Estos conectores tienen los siguientes requisitos:

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

> **Note**
>
> Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Dependencias del tiempo de ejecución del aprendizaje profundo Sagemaker Neo instalado en el dispositivo del núcleo. Para obtener más información, consulte the section called “Instalación de las dependencias del entorno de ejecución de aprendizaje profundo de Neo”.
- Un recurso de ML en el grupo de Greengrass. El recurso de ML debe hacer referencia a un bucket de Amazon S3 que contenga un modelo de detección de objetos. Para obtener más información, consulte Orígenes del modelo Amazon S3.

> **Note**
>
> El modelo debe ser un tipo de modelo de detección de objetos Single Shot Multibox Detector o You Only Look Once v3. Debe compilarse con el compilador de aprendizaje profundo Sagemaker Neo. Para obtener más información, consulte Requisitos del modelo de detección de objetos.

- El conector de Feedback de ML añadido al grupo de Greengrass y configurado. Solo es necesario si desea utilizar el conector para cargar datos de entrada del modelo y publicar predicciones en un tema de MQTT.
- Se requiere el SDK AWS IoT Greengrass de machine learning versión 1.1.0 para interactuar con este conector.

Requisitos del modelo de detección de objetos

Los conectores de Detección de objetosde ML admiten los tipos de modelos de detección de objetos Single Shot multibox Detector (SSD) y You Only Look Once (YOLO) v3. Puede utilizar los componentes de detección de objetos proporcionados por GluonCV para entrenar el modelo con su propio conjunto de datos. También puede utilizar modelos entrenados previamente de GluonCV Model Zoo:

- Modelo SSD entrenado previamente
- Modelo YOLO v3 entrenado previamente

El modelo de detección de objetos debe entrenarse con imágenes de entrada de 512 x 512. Los modelos entrenados previamente de GluonCV Model Zoo ya cumplen este requisito.

Los modelos de detección de objetos entrenados deben compilarse con con el compilador de aprendizaje profundo SageMaker Neo. Al compilar, asegúrese de que el hardware de destino coincida con el hardware del dispositivo de Greengrass Core. Para obtener más información, consulte SageMaker Neo en la Guía para desarrolladores de Amazon SageMaker.

El modelo compilado debe añadirse como un recurso de ML (origen del modelo de Amazon S3) al mismo grupo de Greengrass que el conector.

Parámetros de conector

Estos conectores proporcionan los siguientes parámetros.

`MLModelDestinationPath`

La ruta absoluta al bucket de Amazon S3 que contiene el modelo de ML compatible con Neo. Esta es la ruta de destino que se ha especificado para el recurso del modelo de aprendizaje automático.

Nombre que mostrar en la consola AWS IoT: Ruta de destino del modelo

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`MLModelResourceId`

El ID del recurso de aprendizaje automático que hace referencia al modelo de origen.

Nombre que mostrar en la consola AWS IoT: Recurso de ML del grupo Greengrass

Obligatorio: `true`

Escriba: `S3MachineLearningModelResource`

Patrón válido: `^[a-zA-Z0-9:_-]+$`

`LocalInferenceServiceName`

El nombre del servicio de inferencia local. Las funciones de Lambda definidas por el usuario invocan el servicio pasando el nombre a la función `invoke_inference_service` del SDK

de machine learning de AWS IoT Greengrass. Para ver un ejemplo, consulte the section called "Ejemplo de uso".

Nombre que mostrar en la consola AWS IoT: Nombre del servicio de inferencia local

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

`LocalInferenceServiceTimeoutSeconds`

El tiempo (en segundos) antes de que se termine la solicitud de inferencia. El valor mínimo es 1. El valor predeterminado es 10.

Nombre que mostrar en la consola AWS IoT: Tiempo de espera (segundo)

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[1-9][0-9]*$`

`LocalInferenceServiceMemoryLimitKB`

La cantidad de memoria (en KB) a la que el servicio tiene acceso. El valor mínimo es 1.

Nombre que mostrar en la consola AWS IoT: Límite de memoria

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[1-9][0-9]*$`

`GPUAcceleration`

El contexto de informática de la CPU o GPU (acelerada). Esta propiedad se aplica al solo al conector Image Classification de ML Aarch64 JTX2.

Nombre que mostrar en la consola AWS IoT: Aceleración de la GPU

Obligatorio: `true`

Escriba: `string`

Valores válidos: `CPU` o `GPU`

`MLFeedbackConnectorConfigId`

El ID de la configuración de retroalimentación que se va a utilizar para cargar los datos de entrada del modelo. Debe coincidir con el ID de una configuración de comentarios definida para el conector de comentarios de aprendizaje automático.

Este parámetro solo es necesario si desea utilizar el conector de comentarios de aprendizaje automático para cargar datos de entrada del modelo y publicar predicciones en un tema de MQTT.

Nombre que mostrar en la consola AWS IoT: ID de configuración del conector de ML Feedback connector

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene un conector Object Detection de ML. Este ejemplo crea una instancia del conector Object Detection de ML ARMv7l.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyObjectDetectionConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ObjectDetectionARMv7/versions/1",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "LocalInferenceServiceName": "objectDetection",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "MLFeedbackConnectorConfigId" : "object-detector-random-sampling"
            }
        }
    ]
```

```
}'
```

> **Note**
>
> La función de Lambda de estos conectores tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de Conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Estos conectores aceptan un archivo de imagen como entrada. Los archivos de imagen de entrada deben tener el formato `jpeg` o `png`. Para obtener más información, consulte the section called “Ejemplo de uso”.

Estos conectores no aceptan mensajes MQTT como datos de entrada.

Datos de salida

Estos conectores devuelven una lista con formato de resultados de predicciones para los objetos identificados en la imagen de entrada:

```
    {
        "prediction": [
            [
                14,
                0.9384938478469849,
                0.37763649225234985,
                0.5110225081443787,
                0.6697432398796082,
                0.8544386029243469
            ],
            [
                14,
                0.8859519958496094,
                0,
                0.43536216020584106,
                0.3314110040664673,
                0.9538808465003967
            ],
```

```
            [
                12,
                0.04128098487854004,
                0.5976729989051819,
                0.5747185945510864,
                0.704264223575592,
                0.857937216758728
            ],
            ...
        ]
    }
```

Cada predicción de la lista está incluida entre corchetes y contiene seis valores:

- El primer valor representa la categoría de objeto prevista para el objeto identificado. Las categorías de objetos y sus valores correspondientes se determinan al entrenar el modelo de machine learning de detección de objetos en el compilador de aprendizaje profundo Neo.
- El segundo valor es la puntuación de confianza para la predicción de categorías de objetos. Esto representa la probabilidad de que la predicción sea correcta.
- Los cuatro últimos valores se corresponden con dimensiones de píxeles que representan un cuadro delimitador alrededor del objeto previsto en la imagen.

Estos conectores no publican mensajes MQTT como datos de salida.

Ejemplo de uso

El siguiente ejemplo de función de Lambda utiliza el SDK de Machine Learning AWS IoT Greengrass para interactuar con un conector de detección de objetos de ML.

> **Note**
>
> Puede descargar el SDK desde la página de descargas del SDK de machine learning de AWS IoT Greengrass.

En el ejemplo se inicializa un cliente SDK y de forma sincrónica llama a la función `invoke_inference_service` del SDK para invocar el servicio de inferencia local. Pasa el tipo de algoritmo, el nombre del servicio, el tipo de imagen y contenido de imágenes. A continuación, el ejemplo analiza la respuesta del servicio para obtener los resultados de probabilidad (predicciones).

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='object-detection',
            ServiceName='objectDetection',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}("{}")'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}("{}")'.format(e.__class__.__name__, e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))
    predictions = eval(predictions)

    # Perform business logic that relies on the predictions.

    # Schedule the infer() function to run again in ten second.
    Timer(10, infer).start()
    return

infer()
```

```
def function_handler(event, context):
    return
```

La función `invoke_inference_service` en el SDK de machine learning de AWS IoT Greengrass acepta los argumentos siguientes.

Argumento	Descripción
`AlgoType`	El nombre del tipo de algoritmo que usar para la interferencia. En la actualidad, solo se admite `object-detection` . Obligatorio: `true` Escriba: `string` Valores válidos: `object-detection`
`ServiceName`	El nombre del servicio de inferencia local. Utilice el nombre que especificó para el parámetro `LocalInferenceServiceName` al configurar el conector. Obligatorio: `true` Escriba: `string`
`ContentType`	El tipo mime de la imagen de entrada. Obligatorio: `true` Escriba: `string` Valores válidos: `image/jpeg, image/png`
`Body`	El contenido del archivo de la imagen de entrada. Obligatorio: `true`

Argumento	Descripción
	Escriba: `binary`

Instalación de las dependencias del entorno de ejecución de aprendizaje profundo de Neo en el núcleo de AWS IoT Greengrass

Los conectores de detección de objetos de ML vienen incluidos en el tiempo de ejecución del aprendizaje profundo (DLR) de Sagemaker Neo. Los conectores utilizan el tiempo de ejecución para ofrecer el modelo de ML. Para utilizar estos conectores, debe instalar las dependencias para el DLR en su dispositivo principal.

Antes de instalar las dependencias de DLR, asegúrese de que las bibliotecas del sistema requeridas (con la versiones mínimas especificadas) están presentes en el dispositivo.

NVIDIA Jetson TX2

1. Instalación del conjunto de herramientas CUDA 9.0 y cuDNN 7.0. Puede seguir las instrucciones en the section called “Configuración de otros dispositivos” en el tutorial de Introducción.
2. Habilitar repositorios universales para que el conector pueda instalar software abierto mantenido por la comunidad. Para obtener más información, consulte Repositories/Ubuntu en la documentación de Ubuntu.

   a. Abra el archivo `/etc/apt/sources.list`.

   b. Asegúrese de que las siguientes líneas no tienen comentarios.

   ```
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   ```

3. Guarde una copia del siguiente script de instalación en un archivo llamado `nvidiajtx2.sh` en el dispositivo del núcleo.

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."
   ```

```
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> Si OpenCV no se instala correctamente con este script, puede intentar realizar la compilación desde el código fuente. Para obtener más información, consulte Instalación en Linux en la documentación de OpenCV o vea otros recursos online para su plataforma.

4. En el directorio en el que guardó el archivo, ejecute el siguiente comando:

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Guarde una copia del siguiente script de instalación en un archivo llamado `x86_64.sh` en el dispositivo del núcleo.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
```

```
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
  apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
  echo "OS Release not supported: $release"
  exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> Si OpenCV no se instala correctamente con este script, puede intentar realizar la compilación desde el código fuente. Para obtener más información, consulte Instalación en Linux en la documentación de OpenCV o vea otros recursos online para su plataforma.

2. En el directorio en el que guardó el archivo, ejecute el siguiente comando:

```
sudo x86_64.sh
```

ARMv7 (Raspberry Pi)

1. Guarde una copia del siguiente script de instalación en un archivo llamado `armv7l.sh` en el dispositivo del núcleo.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> Si OpenCV no se instala correctamente con este script, puede intentar realizar la compilación desde el código fuente. Para obtener más información, consulte Instalación en Linux en la documentación de OpenCV o vea otros recursos online para su plataforma.

2. En el directorio en el que guardó el archivo, ejecute el siguiente comando:

```
sudo bash armv7l.sh
```

> **Note**
>
> En una solución Raspberry Pi, el uso de `pip` para instalar dependencias de aprendizaje automático es una operación de uso intensivo de memoria que puede provocar que el dispositivo se quede sin memoria y deje de responder. Como alternativa, puede aumentar temporalmente el tamaño de intercambio. En `/etc/dphys-swapfile`, aumente el valor de la variable `CONF_SWAPSIZE` y, a continuación, ejecute el siguiente comando para reiniciar `dphys-swapfile`.

```
/etc/init.d/dphys-swapfile restart
```

Registro y solución de problemas

En función de su configuración del grupo, los registros de evento y error se escriben en el registro de CloudWatch, el sistema de archivos local o ambos. Los registros de este conector utilizan el prefijo `LocalInferenceServiceName`. Si el conector se comporta de forma inesperada, compruebe los registros del conector. Estos suelen contener información de depuración útil, como, por ejemplo, que falta una dependencia de biblioteca de ML o la causa de un error de inicio del conector.

Si el grupo AWS IoT Greengrass está configurado para escribir registros locales, el conector escribe los archivos de registro en *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/aws/`. Para obtener más información sobre los registros de Greengrass, consulte the section called “Monitorización con registros de AWS IoT Greengrass”.

Utilice la siguiente información como ayuda para solucionar problemas con los conectores de la Detection de objetos de ML.

Bibliotecas del sistema obligatorias

Las siguientes pestañas muestran las bibliotecas del sistema necesarias para cada conector Object Detection de ML.

ML Object Detection Aarch64 JTX2

Library	Versión mínima
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	no aplicable
libcudart.so.9.0	no aplicable
libcudnn.so.7	no aplicable
libcufft.so.9.0	no aplicable

Library	Versión mínima
libcurand.so.9.0	no aplicable
libcusolver.so.9.0	no aplicable
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libnvinfer.so.4	no aplicable
libnvrm_gpu.so	no aplicable
libnvrm.so	no aplicable
libnvidia-fatbinaryloader.so.28.2.1	no aplicable
libnvos.so	no aplicable
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Object Detection x86_64

Library	Versión mínima
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23

Library	Versión mínima
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Object Detection ARMv7

Library	Versión mínima
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problemas

Síntoma	Solución
En una Raspberry Pi, se registra el siguiente mensaje de error y no está utilizando la cámara: `Failed to initialize libdc1394`	Ejecute el comando siguiente para deshabilitar el controlador: `sudo ln /dev/null /dev/raw1394`

Síntoma	Solución
	Esta operación es efímera. El enlace simbólico desaparece después de reiniciar. Consulte el manual de su distribución de SO para obtener información sobre cómo crear automáticamente el enlace al reiniciar.

Licencias

Los conectores Object Detection de ML incluyen las siguientes licencias y software de terceros:

- AWS SDK for Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

- Tiempo de ejecución de aprendizaje profundo/Licencia Apache 2.0
- six/MIT

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- Cómo realizar la inferencia de machine learning
- Algoritmo de detección de objetos en la Guía para desarrolladores de Amazon SageMaker

Conector adaptador de protocolo Modbus-RTU

El conector del adaptador de protocolo Modbus-RTU recopila información de los dispositivos Modbus RTU que están en el grupo AWS IoT Greengrass.

Este conector recibe los parámetros de una solicitud de Modbus RTU de una función de Lambda definida por el usuario. Envía la solicitud correspondiente y, a continuación, publica la respuesta del dispositivo de destino como mensaje de MQTT.

Este conector tiene las siguientes versiones.

Versión	ARN
3	`arn:aws:greengrass:  region::/connectors/ModbusRTUProtocolAdapter/versions/3`
2	`arn:aws:greengrass:  region::/connectors/ModbusRTUProtocolAdapter/versions/2`
1	`arn:aws:greengrass:  region::/connectors/ModbusRTUProtocolAdapter/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

> **Note**
>
> Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Una conexión física entre los dispositivos de Modbus y el núcleo AWS IoT Greengrass. El núcleo debe ser conectado físicamente a la red de Modbus RTU a través de un puerto serie (por ejemplo, un puerto USB).
- Un recurso de dispositivo local en el grupo Greengrass que apunta al puerto serie Modbus físico.
- Una función de Lambda definida por el usuario que envía los parámetros de una solicitud de Modbus RTU a este conector. Los parámetros de solicitud deben cumplir los patrones esperados e incluir os ID y las direcciones de los dispositivos de destino en la red de Modbus RTU. Para obtener más información, consulte the section called “Datos de entrada”.

Versions 1 - 2

- Software AWS IoT Greengrass Core versión 1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Una conexión física entre los dispositivos de Modbus y el núcleo AWS IoT Greengrass. El núcleo debe ser conectado físicamente a la red de Modbus RTU a través de un puerto serie (por ejemplo, un puerto USB).
- Un recurso de dispositivo local en el grupo Greengrass que apunta al puerto serie Modbus físico.
- Una función de Lambda definida por el usuario que envía los parámetros de una solicitud de Modbus RTU a este conector. Los parámetros de solicitud deben cumplir los patrones esperados e incluir os ID y las direcciones de los dispositivos de destino en la red de Modbus RTU. Para obtener más información, consulte the section called “Datos de entrada”.

Parámetros de conector

Este conector admite los siguientes parámetros:

`ModbusSerialPort-ResourceId`

El ID del recurso de dispositivo local que representa el puerto de serie físico de Modbus.

> **Note**
>
> A este conector se le concede acceso de lectura y escritura al recurso.

Nombre para mostrar en la consola AWS IoT: Recurso de puerto serie Modbus

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`ModbusSerialPort`

La ruta absoluta del puerto de serie físico de Modbus del dispositivo. Esta es la ruta de origen que haya especificado para el recurso de dispositivo local de Modbus.

Nombre para mostrar en la consola AWS IoT: Ruta de origen del recurso del puerto serie Modbus

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando CLI crea un `ConnectorDefinition` con una versión inicial que contiene el conector del adaptador del protocolo Modbus-RTU.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
```

```
            "Id": "MyModbusRTUProtocolAdapterConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ModbusRTUProtocolAdapter/versions/3",
            "Parameters": {
                "ModbusSerialPort-ResourceId": "MyLocalModbusSerialPort",
                "ModbusSerialPort": "/path-to-port"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

> **Note**
>
> Después de implementar el conector del adaptador del protocolo Modbus-RTU, puede utilizar AWS IoT Things Graph para organizar las interacciones entre los dispositivos del grupo. Para obtener más información, consulte Modbus en la Guía del usuario de AWS IoT Things Graph.

Datos de entrada

Este conector acepta los parámetros de una solicitud de Modbus RTU de una función de Lambda definida por el usuario en un tema de MQTT. Los mensajes de entrada deben tener un formato JSON válido.

Filtro de temas en la suscripción

`modbus/adapter/request`

Propiedades de mensajes

El mensaje de solicitud varía en función del tipo de solicitud de Modbus RTU que representa. Las siguientes propiedades son necesarias para todas las solicitudes:

- En el objeto `request`:
  - `operation`. Nombre de la operación que se va a ejecutar. Por ejemplo, especifique `"operation": "ReadCoilsRequest"` para leer salidas digitales (coils). Este valor debe ser una cadena Unicode. Para obtener información sobre las operaciones admitidas, consulte the section called “Solicitudes y respuestas de Modbus RTU”.
  - `device`. El dispositivo de destino de la solicitud. Este valor debe estar entre `0 - 247`.
- La propiedad `id`. ID de la solicitud. Este valor se utiliza para la desduplicación de datos y se devuelve tal cual se encuentra en la propiedad `id` de todas las respuestas, incluidas las de error. Este valor debe ser una cadena Unicode.

> **Note**
>
> Si la solicitud incluye un campo de dirección, debe especificar el valor como un entero. Por ejemplo, `"address": 1`.

El resto de los parámetros que se incluirán en la solicitud dependen de la operación. Todos los parámetros de solicitud son necesarios, excepto el CRC, que se gestiona por separado. Para ver ejemplos, consulte the section called “Solicitudes y respuestas de ejemplo”.

Ejemplo de entrada: Solicitud de lectura de salidas digitales (coils)

```
{
    "request": {
        "operation": "ReadCoilsRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

Datos de salida

Este conector publica las respuestas a las solicitudes de Modbus RTU entrantes.

Filtro de temas en la suscripción

```
modbus/adapter/response
```

Propiedades de mensajes

El formato del mensaje de respuesta varía en función de la solicitud correspondiente y el estado de la respuesta. Para ver ejemplos, consulte the section called “Solicitudes y respuestas de ejemplo”.

> **Note**
>
> Una respuesta para una operación de escritura es simplemente un eco de la solicitud. Aunque no se devuelve información significante para las respuestas de escritura, se recomienda comprobar el estado de la respuesta.

Cada respuesta incluye las siguientes propiedades:

- En el objeto `response`:
  - `status`. El estado de la solicitud. El estado puede ser uno de los siguientes valores:
    - `Success`. La solicitud, válida, se envía a la red de Modbus RTU y se devuelve una respuesta.
    - `Exception`. La solicitud, válida, se envía a la red de Modbus RTU y se devuelve una respuesta de excepción. Para obtener más información, consulte the section called “Estado de respuesta: excepción”.
    - No `Response`. La solicitud no era válida y el conector detecta el error antes de que la solicitud se envíe a través de la red de Modbus RTU. Para obtener más información, consulte the section called “Estado de respuesta: sin respuesta”.
  - `device`. El dispositivo al que se envía la solicitud.
  - `operation`. El tipo de solicitud que se envía.
  - `payload`. El contenido de la respuesta que se ha devuelto. Si `status` es `No Response`, este objeto contiene únicamente una propiedad `error` con la descripción de error (por ejemplo, `"error": "[Input/Output] No Response received from the remote unit"`).
- La propiedad `id`. La ID de la solicitud, que se utilizan para la desduplicación de datos.

Ejemplo de salida: Correcto

```
{
    "response" : {
        "status" : "success",
```

```
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 1,
         "bits": [1]
     }
     },
     "id" : "TestRequest"
}
```

Ejemplo de salida: Error

```
{
    "response" : {
        "status" : "fail",
        "error_message": "Internal Error",
        "error": "Exception",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 129,
         "exception_code": 2
     }
     },
     "id" : "TestRequest"
}
```

Para obtener más ejemplos, consulte the section called “Solicitudes y respuestas de ejemplo”.

Solicitudes y respuestas de Modbus RTU

Este conector acepta los parámetros de solicitud de Modbus RTU como datos de entrada y publica las respuestas como datos de salida.

Se admiten las siguientes operaciones comunes.

Nombre de la operación en la solicitud	Código de característica en la respuesta
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02

Nombre de la operación en la solicitud	Código de característica en la respuesta
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

Solicitudes y respuestas de ejemplo

A continuación, se muestran ejemplos de solicitudes y respuestas de las operaciones compatibles.

Leer bobinas

Ejemplo de solicitud:

```
{
    "request": {
        "operation": "ReadCoilsRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
    "response": {
        "status": "success",
        "device": 1,
```

```
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 1,
         "bits": [1]
     }
     },
     "id" : "TestRequest"
}
```

Leer entradas discretas

Ejemplo de solicitud:

```
{
    "request": {
        "operation": "ReadDiscreteInputsRequest",
        "device": 1,
        "address": 1,
        "count": 1
    },
    "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
    "response": {
        "status": "success",
        "device": 1,
        "operation": "ReadDiscreteInputsRequest",
        "payload": {
            "function_code": 2,
            "bits": [1]
        }
     },
     "id" : "TestRequest"
}
```

Leer registros mantenidos

Ejemplo de solicitud:

```
{
```

```
    "request": {
        "operation": "ReadHoldingRegistersRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "ReadHoldingRegistersRequest",
     "payload": {
         "function_code": 3,
            "registers": [20,30]
     }
     },
     "id" : "TestRequest"
}
```

Leer registros de entrada

Ejemplo de solicitud:

```
{
    "request": {
        "operation": "ReadInputRegistersRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

Escribir una única bobina

Ejemplo de solicitud:

```
{
```

```
    "request": {
        "operation": "WriteSingleCoilRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "WriteSingleCoilRequest",
     "payload": {
         "function_code": 5,
         "address": 1,
         "value": true
     }
     },
     "id" : "TestRequest"
```

Escribir un único registro

Ejemplo de solicitud:

```
{
    "request": {
        "operation": "WriteSingleRegisterRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

Escribir varias bobinas

Ejemplo de solicitud:

```
{
```

```
    "request": {
        "operation": "WriteMultipleCoilsRequest",
     "device": 1,
     "address": 1,
     "values": [1,0,0,1]
    },
    "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "WriteMultipleCoilsRequest",
     "payload": {
         "function_code": 15,
         "address": 1,
         "count": 4
     }
     },
     "id" : "TestRequest"
}
```

Escribir varios registros

Ejemplo de solicitud:

```
{
    "request": {
        "operation": "WriteMultipleRegistersRequest",
     "device": 1,
     "address": 1,
     "values": [20,30,10]
    },
    "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
    "response": {
```

```
        "status": "success",
        "device": 1,
     "operation": "WriteMultipleRegistersRequest",
     "payload": {
         "function_code": 23,
         "address": 1,
         "count": 3
     }
     },
     "id" : "TestRequest"
}
```

Máscara de registro de escritura

Ejemplo de solicitud:

```
{
    "request": {
        "operation": "MaskWriteRegisterRequest",
     "device": 1,
     "address": 1,
        "and_mask": 175,
        "or_mask": 1
    },
    "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "MaskWriteRegisterRequest",
     "payload": {
         "function_code": 22,
            "and_mask": 0,
            "or_mask": 8
     }
     },
     "id" : "TestRequest"
}
```

Leer y escribir varios registros

Ejemplo de solicitud:

```
{
    "request": {
        "operation": "ReadWriteMultipleRegistersRequest",
     "device": 1,
     "read_address": 1,
        "read_count": 2,
        "write_address": 3,
        "write_registers": [20,30,40]
    },
    "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "ReadWriteMultipleRegistersRequest",
     "payload": {
         "function_code": 23,
         "registers": [10,20,10,20]
     }
     },
     "id" : "TestRequest"
}
```

> **Note**
>
> Los registros devueltos en esta respuesta son los registros desde los que se lee.

Estado de respuesta: excepción

Las excepciones pueden producirse cuando el formato de la solicitud es válido, pero la solicitud no se completó correctamente. En este caso, la respuesta contiene la siguiente información:

- `status` se establece en `Exception`.

- `function_code` equivale al código de la característica de la solicitud + 128.
- `exception_code` contiene el código de excepción. Para obtener más información, consulte los códigos de excepción de Modbus.

Ejemplo:

```
{
    "response" : {
        "status" : "fail",
        "error_message": "Internal Error",
        "error": "Exception",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 129,
         "exception_code": 2
     }
     },
     "id" : "TestRequest"
}
```

Estado de respuesta: sin respuesta

Este conector realiza las comprobaciones de validación de la solicitud de Modbus. Por ejemplo, comprueba los formatos no válidos y los campos que faltan. Si no se supera la validación, el conector no envía la solicitud. En su lugar, devuelve una respuesta que contiene la siguiente información:

- `status` se establece en `No Response`.
- `error` contiene el motivo del error.
- `error_message` contiene el mensaje de error.

Ejemplos:

```
{
    "response" : {
        "status" : "fail",
        "error_message": "Invalid address field. Expected <type 'int'>, got <type
 'str'>",
        "error": "No Response",
```

```
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "error": "Invalid address field. Expected <type 'int'>, got <type 'str'>"
     }
     },
     "id" : "TestRequest"
}
```

Si la solicitud selecciona como destino un dispositivo inexistente o si la red de Modbus RTU no funciona, es posible que aparezca una respuesta `ModbusIOException`, que utiliza el formato de respuesta No.

```
{
    "response" : {
        "status" : "fail",
        "error_message": "[Input/Output] No Response received from the remote unit",
        "error": "No Response",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "error": "[Input/Output] No Response received from the remote unit"
     }
     },
     "id" : "TestRequest"
}
```

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.
> - Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de Core AWS IoT Greengrass para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.
3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el recurso de dispositivo local requerido y conceda acceso de lectura/escritura a la función de Lambda.

   c. Agregue el conector y configure sus parámetros.

   d. Agregue suscripciones que permitan al conector recibir datos de entrada y enviar datos de salida en filtros de tema compatibles.

      - Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
      - Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.
4. Implemente el grupo.
5. En la consola de AWS IoT, en la página Prueba suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

   Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la característica envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector.

```
import greengrasssdk
import json

TOPIC_REQUEST = 'modbus/adapter/request'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_read_coils_request():
 request = {
  "request": {
   "operation": "ReadCoilsRequest",
   "device": 1,
   "address": 1,
   "count": 1
  },
  "id": "TestRequest"
 }
 return request

def publish_basic_request():
 iot_client.publish(payload=json.dumps(create_read_coils_request()),
 topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
 return
```

Licencias

El conector del adaptador del protocolo Modbus-RTU incluye las siguientes licencias y software de terceros:

- pymodbus/BSD
- pyserial/BSD

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se actualizó el conector ARN para el soporte de Región de AWS. Se ha mejorado el registro de errores.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”

Conector adaptador de protocolo Modbus-TCP

El conector del adaptador de protocolo Modbus-TCP recopila datos de los dispositivos locales a través del protocolo Modbus-TCP y los publica en las secuencias `StreamManager` seleccionadas.

También puede usar este conector con el conector IoT SiteWise y tu puerta de enlace IoT SiteWise. Su puerta de enlace debe proporcionar la configuración del conector. Para obtener más información, consulte Configurar una fuente Modbus TCP en la guía del usuario de IoT SiteWise.

> **Note**
>
> Este conector se ejecuta en modo sin aislamiento de contenedores, por lo que puede implementarlo en un grupo AWS IoT Greengrass que se ejecute en un contenedor de Docker.

Este conector tiene las siguientes versiones.

Versión	ARN
3	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/3`
2	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/2`
1	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 1 - 3

- Software AWS IoT Greengrass Core versión 1.10.2 o posterior.
- Administrador de secuencias habilitado en el grupo de AWS IoT Greengrass.
- Java 8 instalado en el dispositivo de núcleo y añadido a la variable de entorno `PATH`.

> **Note**
>
> Este conector solo está disponible en las siguientes regiones:
>
> - ap-southeast-1
> - ap-southeast-2
> - eu-central-1
> - eu-west-1
> - us-east-1

- us-west-2
- cn-north-1

Parámetros de conector

Este conector admite los siguientes parámetros:

`LocalStoragePath`

El directorio del host de AWS IoT Greengrass en la que el conector de IoT SiteWise puede escribir datos persistentes. El directorio predeterminado es `/var/sitewise`.

Nombre para mostrar en la consola AWS IoT: Ruta de almacenamiento local

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^\s*$|\/.`

`MaximumBufferSize`

El tamaño máximo en GB para el uso del disco de IoT SiteWise. El tamaño por defecto es de 10 GB.

Nombre para mostrar en la consola AWS IoT: Tamaño máximo del búfer de disco

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^\s*$|[0-9]+`

`CapabilityConfiguration`

El conjunto de configuraciones de recopiladores Modbus TCP desde las que el conector recopila datos o a las que se conecta.

Nombre para mostrar en la consola AWS IoT: CapabilityConfiguration

Obligatorio: `false`

Tipo: una cadena JSON bien formada que define el conjunto de configuraciones de comentarios admitidas.

A continuación se muestra un ejemplo de una `CapabilityConfiguration`.

```
{
    "sources": [
        {
            "type": "ModBusTCPSource",
            "name": "SourceName1",
            "measurementDataStreamPrefix": "SourceName1_Prefix",
            "destination": {
                "type": "StreamManager",
                "streamName": "SiteWise_Stream_1",
                "streamBufferSize": 8
            },
            "endpoint": {
                "ipAddress": "127.0.0.1",
                "port": 8081,
                "unitId": 1
            },
            "propertyGroups": [
                {
                    "name": "GroupName",
                    "tagPathDefinitions": [
                        {
                            "type": "ModBusTCPAddress",
                            "tag": "TT-001",
                            "address": "30001",
                            "size": 2,
                            "srcDataType": "float",
                            "transformation": "byteWordSwap",
                            "dstDataType": "double"
                        }
                    ],
                    "scanMode": {
                        "type": "POLL",
                        "rate": 100
                    }
                }
            ]
        }
    ]
```

```
}
```

Ejemplo de creación de conector (AWS CLI)

El siguiente comando CLI crea un `ConnectorDefinition` con una versión inicial que contiene el conector del adaptador del protocolo Modbus-TCP.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '
{
    "Connectors": [
        {
            "Id": "MyModbusTCPConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/ModbusTCP/
versions/3",
            "Parameters": {
                "capability_configuration": "{\"version\":1,\"namespace\":
\"iotsitewise:modbuscollector:1\",\"configuration\":\"{\"sources\":[{\"type
\":\"ModBusTCPSource\",\"name\":\"SourceName1\",\"measurementDataStreamPrefix
\":\"\",\"endpoint\":{\"ipAddress\":\"127.0.0.1\",\"port\":8081,\"unitId\":1},
\"propertyGroups\":[{\"name\":\"PropertyGroupName\",\"tagPathDefinitions\":[{\"type
\":\"ModBusTCPAddress\",\"tag\":\"TT-001\",\"address\":\"30001\",\"size\":2,
\"srcDataType\":\"hexdump\",\"transformation\":\"noSwap\",\"dstDataType\":\"string
\"}],\"scanMode\":{\"rate\":200,\"type\":\"POLL\"}}],\"destination\":{\"type\":
\"StreamManager\",\"streamName\":\"SiteWise_Stream\",\"streamBufferSize\":10},
\"minimumInterRequestDuration\":200}]}\"}"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

Datos de entrada

Este conector no acepta mensajes MQTT como datos de entrada.

Datos de salida

Este conector publica datos en `StreamManager`. Debe configurar el flujo de mensajes de destino. Los mensajes de salida tienen la siguiente estructura:

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

Licencias

El conector adaptador del protocolo Modbus-TCP incluye el siguiente software/licencia de terceros:

- Petri Modbus digital

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios	Fecha
3 (recomendado)	Esta versión contiene correcciones de errores.	22 de diciembre de 2021
2	Se agregó soporte para cadenas fuente codificadas en ASCII, UTF8 e ISO8859.	24 de mayo de 2021
1	Versión inicial.	15 de diciembre de 2020

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”

Conector Raspberry Pi GPIO

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

El conector GPIO de Raspberry Pi controla los pines de entrada/salida de uso general (GPIO) en un dispositivo del núcleo de Raspberry Pi.

Este conector sondea pines de entrada a un intervalo especificado y publica los cambios de estado en temas de MQTT. También acepta las solicitudes de lectura y escritura como mensajes de MQTT de las funciones de Lambda definidas por el usuario. Las solicitudes de escritura se utilizan para establecer el pin en tensión alta o baja.

El conector proporciona los parámetros que se usan para designar los pines de entrada y salida. Este comportamiento se configura antes de la implementación del grupo. No se puede cambiar en el tiempo de ejecución.

- Los pines de entrada se pueden utilizar para recibir datos de dispositivos periféricos.
- Los pines de salida se pueden utilizar para controlar periféricos o enviar datos a periféricos.

Puede utilizar este conector para admitir muchas situaciones, como:

- Controlar las luces LED verde, ámbar y roja de un semáforo.

- Controlar un ventilador (conectado a un relé eléctrico) en función de los datos de un sensor de humedad.
- Alertar a los empleados de una tienda minorista cuando los clientes pulsen un botón.
- Usar un interruptor de luz inteligente para controlar otros dispositivos de IoT.

> **Note**
>
> Este conector no es adecuado para aplicaciones que tienen requisitos en tiempo real. Los eventos con períodos cortos podrían pasarse por alto.

Este conector tiene las siguientes versiones.

Versión	ARN
3	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/3`
2.	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/2`
1	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.

- Versión .7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Raspberry Pi 4 modelo B o Raspberry Pi 3 modelo B/B+. Debe conocer la secuencia de pines de su Raspberry Pi. Para obtener más información, consulte the section called “Secuencia de pines de GPIO”.
- Un recurso de dispositivo local en el grupo de Greengrass que apunta a `/dev/gpiomem` en la Raspberry Pi. Si crea el recurso en la consola, debe seleccionar la opción Añadir automáticamente los permisos de grupo de OS del grupo de Linux propietario del recurso. En la API, defina la propiedad `GroupOwnerSetting.AutoAddGroupOwner` en `true`.
- El módulo RPi.GPIO instalado en el dispositivo Raspberry Pi. En Raspbian, este módulo se instala de forma predeterminada. Puede utilizar el comando siguiente para volverlo a instalar:

  ```
  sudo pip install RPi.GPIO
  ```

Versions 1 - 2

- Software AWS IoT Greengrass Core versión 1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Raspberry Pi 4 modelo B o Raspberry Pi 3 modelo B/B+. Debe conocer la secuencia de pines de su Raspberry Pi. Para obtener más información, consulte the section called “Secuencia de pines de GPIO”.
- Un recurso de dispositivo local en el grupo de Greengrass que apunta a `/dev/gpiomem` en la Raspberry Pi. Si crea el recurso en la consola, debe seleccionar la opción Añadir automáticamente los permisos de grupo de OS del grupo de Linux propietario del recurso. En la API, defina la propiedad `GroupOwnerSetting.AutoAddGroupOwner` en `true`.
- El módulo RPi.GPIO instalado en el dispositivo Raspberry Pi. En Raspbian, este módulo se instala de forma predeterminada. Puede utilizar el comando siguiente para volverlo a instalar:

  ```
  sudo pip install RPi.GPIO
  ```

Secuencia de pines de GPIO

El conector Raspberry Pi GPIO hace referencia a los pines GPIO por el esquema de numeración del sistema en chip (SoC) subyacente, no por la disposición física de los pines GPIO. El orden físico de

los pines puede variar en las versiones de Raspberry Pi. Para obtener más información, consulte GPIO en la documentación de Raspberry Pi.

El conector no puede validar que los pines de entrada y salida configurados se asignen correctamente en el hardware subyacente de su Raspberry Pi. Si la configuración de pines no es válida, el conector devuelve un error de tiempo de ejecución cuando se intenta iniciar en el dispositivo. Para solucionar este problema, vuelva a configurar el conector y, a continuación, impleméntelo de nuevo.

> **Note**
>
> Asegúrese de que los periféricos para los pines de GPIO están correctamente cableados para evitar daños de componentes.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

`InputGpios`

Lista separada por comas de números de pines de GPIO para configurar como entradas. Si lo desea, añada `U` para establecer una resistencia de subida de pines o `D` para establecer la resistencia de bajada. Ejemplo: `"5,6U,7D"`.

Nombre para mostrar en la consola AWS IoT: Input GPIO pins

Obligatorio: `false`. Debe especificar pines de entrada, de salida o ambos.

Escriba: `string`

Patrón válido: `^$|^[0-9]+[UD]?(,[0-9]+[UD]?)*$`

`InputPollPeriod`

El intervalo (en milisegundos) entre cada operación de sondeo, que comprueba los pines de GPIO de entrada para buscar cambios de estado. El valor mínimo es 1.

Este valor depende del escenario y del tipo de dispositivos que se sondearán. Por ejemplo, un valor de `50` debe ser lo suficientemente rápido para detectar una pulsación de botón.

Nombre para mostrar en la consola AWS IoT: Pines GPIO de entrada

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^$|^[1-9][0-9]*$`

`OutputGpios`

Lista separada por comas de números de pines de GPIO para configurar como salidas. Si lo desea, añada `H` para establecer un estado alto (1) o `L` para establecer un estado bajo (0). Ejemplo: `"8H,9,27L"`.

Nombre para mostrar en la consola AWS IoT: Pines GPIO de salida

Obligatorio: `false`. Debe especificar pines de entrada, de salida o ambos.

Escriba: `string`

Patrón válido: `^$|^[0-9]+[HL]?(,[0-9]+[HL]?)*$`

`GpioMem-ResourceId`

El ID del recurso de dispositivo local que representa `/dev/gpiomem`.

> **Note**
>
> A este conector se le concede acceso de lectura y escritura al recurso.

Nombre para mostrar en la consola AWS IoT: Resource for /dev/gpiomem device

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector GPIO de Raspberry Pi.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyRaspberryPiGPIOConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/
versions/3",
            "Parameters": {
                "GpioMem-ResourceId": "my-gpio-resource",
                "InputGpios": "5,6U,7D",
                "InputPollPeriod": 50,
                "OutputGpios": "8H,9,27L"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de Conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta solicitudes de lectura o escritura para pines de GPIO en dos temas de MQTT.

- Solicitudes de lectura en el tema `gpio/+/+/read`.
- Solicitudes de escritura en el tema `gpio/+/+/write`.

Para publicar en estos temas, reemplace los comodines + con el nombre de cosa de núcleo y el número de pines de destino, respectivamente. Por ejemplo:

```
gpio/core-thing-name/gpio-number/read
```

> **Note**
>
> Actualmente, cuando crea una suscripción que utiliza el conector GPIO de Raspberry Pi, debe especificar un valor para al menos uno de los comodines + en el tema.

Filtro de temas: `gpio/+/+/read`

Use este tema para que el conector lea el estado del pin de GPIO que se especifica en el tema.

El conector publica la respuesta en el tema de salida correspondiente (por ejemplo, `gpio/`*`core-thing-name`*`/`*`gpio-number`*`/state`).

Propiedades de mensajes

Ninguno. Los mensajes que se envían a este tema se pasan por alto.

Filtro de temas: `gpio/+/+/write`

Use este tema para enviar solicitudes de escritura a un pin de GPIO. Esto indica al conector que establezca el pin de GPIO que se especifica en el tema con una tensión baja o alta.

- `0` establece el pin en voltaje bajo.
- `1` establece el pin en voltaje alto.

El conector publica la respuesta en el tema de salida `/state` correspondiente (por ejemplo, `gpio/`*`core-thing-name`*`/`*`gpio-number`*`/state`).

Propiedades de mensajes

El valor `0` o `1`, como un número entero o cadena.

Ejemplo de entrada

```
0
```

Datos de salida

Este conector publica los datos en dos temas:

- Los cambios de estado de alto o bajo en el tema `gpio/+/+/state`.
- Los errores en el tema `gpio/+/error`.

Filtro de temas: `gpio/+/+/state`

Use este tema para escuchar los cambios de estado de pines de entrada y respuestas de las solicitudes de lectura. El conector devuelve la cadena `"0"` si el pin se encuentra en un estado bajo, o `"1"` si se trata de un estado alto.

Al publicar en este tema, el conector reemplaza los comodines + por el nombre de cosa de núcleo y el pin de destino, respectivamente. Por ejemplo:

```
gpio/core-thing-name/gpio-number/state
```

> **Note**
>
> Actualmente, cuando crea una suscripción que utiliza el conector GPIO de Raspberry Pi, debe especificar un valor para al menos uno de los comodines + en el tema.

Ejemplo de salida

```
0
```

Filtro de temas: `gpio/+/error`

Use este tema para escuchar errores. El conector publica en este tema como resultado de una solicitud no válida (por ejemplo, cuando se solicita un cambio de estado en un pin de entrada).

Al publicar en este tema, el conector reemplaza el comodín + por el nombre de cosa de núcleo.

Ejemplo de salida

```
{
   "topic": "gpio/my-core-thing/22/write",
   "error": "Invalid GPIO operation",
   "long_description": "GPIO 22 is configured as an INPUT GPIO. Write operations
 are not permitted."
 }
```

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.
> - Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de AWS IoT Greengrass Core para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.
3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el recurso de dispositivo local requerido y conceda acceso de lectura/escritura a la función de Lambda.

   c. Agregue el conector y configure sus parámetros.

   d. Agregue suscripciones que permitan al conector recibir datos de entrada y enviar datos de salida en filtros de tema compatibles.

      - Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.

- Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.

4. Implemente el grupo.
5. En la consola de AWS IoT, en la página Prueba, suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

   Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la función envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector. En este ejemplo, se envían solicitudes de lectura de un conjunto de pins de GPIO de entrada. En él se muestra cómo crear temas utilizando el nombre de objeto y el número pin del dispositivo central.

```
import greengrasssdk
import json
import os

iot_client = greengrasssdk.client('iot-data')
INPUT_GPIOS = [6, 17, 22]

thingName = os.environ['AWS_IOT_THING_NAME']

def get_read_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'read'])

def get_write_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'write'])

def send_message_to_connector(topic, message=''):
    iot_client.publish(topic=topic, payload=str(message))

def set_gpio_state(gpio, state):
    send_message_to_connector(get_write_topic(gpio), str(state))

def read_gpio_state(gpio):
```

```
    send_message_to_connector(get_read_topic(gpio))

def publish_basic_message():
    for i in INPUT_GPIOS:
     read_gpio_state(i)

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licencias

El conector GPIO de Raspberry Pi incluye las siguientes licencias y software de terceros:

- RPi.GPIO/MIT

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se actualizó el conector ARN para el soporte de Región de AWS.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- GPIO en la documentación de Raspberry Pi

Conector Serial Stream

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

El conector Serial Stream realiza las operaciones de lectura y escritura en un puerto de serie de un dispositivo de núcleo de AWS IoT Greengrass.

Este conector es compatible con dos modos de operación:

- Lectura bajo demanda. Recibe las solicitudes de lectura y escritura en temas de MQTT y publica la respuesta de la operación de lectura o el estado de la operación de escritura.
- Lectura de sondeo. Lee desde el puerto de serie a intervalos regulares. Este modo también admite las solicitudes de lectura bajo demanda.

> **Note**
>
> Las solicitudes de lectura se limitan a una longitud de lectura máxima de 63994 bytes. Las solicitudes de escritura se limitan a una longitud de datos máxima de 128000 bytes.

Este conector tiene las siguientes versiones.

Versión	ARN
3	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/3`
2	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/2`
1	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

  > **Note**
  >
  > Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Un recurso de dispositivo local en el grupo Greengrass que apunta al puerto serie de destino.

> **Note**
>
> Antes de implementar este conector, le recomendamos que configure el puerto de serie y verifique que puede leer y escribir en él.

Versions 1 - 2

- Software AWS IoT Greengrass Core versión 1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Un recurso de dispositivo local en el grupo Greengrass que apunta al puerto serie de destino.

> **Note**
>
> Antes de implementar este conector, le recomendamos que configure el puerto de serie y verifique que puede leer y escribir en él.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

`BaudRate`

La velocidad en baudios de la conexión de serie.

Nombre que mostrar en la consola AWS IoT: Velocidad en baudios

Obligatorio: `true`

Escriba: `string`

Valores válidos: `110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400`

Patrón válido: `^110$|^300$|^600$|^1200$|^2400$|^4800$|^9600$|^14400$|^19200$|^28800$|^38400$|^56000$|^57600$|^115200$|^230400$`

`Timeout`

El tiempo de espera (en segundos) para una operación de lectura.

Nombre que mostrar en la consola AWS IoT: Tiempo de espera

Obligatorio: `true`

Escriba: `string`

Valores válidos: `1 - 59`

Patrón válido: `^([1-9]|[1-5][0-9])$`

`SerialPort`

La ruta absoluta al puerto de serie físico del dispositivo. Esta es la ruta de origen que haya especificado para el recurso de dispositivo local.

Nombre que mostrar en la consola AWS IoT: Puerto de serie

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[/a-zA-Z0-9_-]+`

`SerialPort-ResourceId`

El ID del recurso de dispositivo local que representa el puerto de serie físico.

> **Note**
>
> A este conector se le concede acceso de lectura y escritura al recurso.

Nombre para mostrar en la consola AWS IoT: Recurso de puerto de serie

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[a-zA-Z0-9_-]+`

`PollingRead`

Establece el modo de lectura: lectura de sondeo o lectura bajo demanda.

- Para el modo de lectura de sondeo, especifique `true`. En este modo, se necesitan las propiedades `PollingInterval`, `PollingReadType` y `PollingReadLength`.
- Para el modo de lectura bajo demanda, especifique `false`. En este modo, los valores de tipo y longitud se especifican en la solicitud de lectura.

Nombre que mostrar en la consola AWS IoT: Modo de lectura

Obligatorio: `true`

Escriba: `string`

Valores válidos: `true, false`

Patrón válido: `^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`PollingReadLength`

La longitud de los datos (en bytes) que se leen en cada operación de lectura de sondeo. Esto se aplica únicamente cuando se utiliza de modo de lectura de sondeo.

Nombre para mostrar en la consola AWS IoT: Longitud de lectura del sondeo

Obligatorio: `false`. Esta propiedad es obligatoria cuando `PollingRead` es `true`.

Escriba: `string`

Patrón válido: `^(|[1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-2][0-9]{3}|63[0-8][0-9]{2}|639[0-8][0-9]|6399[0-4])$`

`PollingReadInterval`

El intervalo (en segundos) en el que tiene lugar la lectura de sondeo. Esto se aplica únicamente cuando se utiliza de modo de lectura de sondeo.

Nombre para mostrar en la consola AWS IoT: Intervalo de lectura del sondeo

Obligatorio: `false`. Esta propiedad es obligatoria cuando `PollingRead` es `true`.

Escriba: `string`

Valores válidos: 1 - 999

Patrón válido: `^(|[1-9]|[1-9][0-9]|[1-9][0-9][0-9])$`

`PollingReadType`

El tipo de datos que el subproceso de sondeo lee. Esto se aplica únicamente cuando se utiliza de modo de lectura de sondeo.

Nombre para mostrar en la consola AWS IoT: Tipo de lectura de sondeo

Obligatorio: `false`. Esta propiedad es obligatoria cuando `PollingRead` es `true`.

Escriba: `string`

Valores válidos: `ascii, hex`

Patrón válido: `^(|[Aa][Ss][Cc][Ii][Ii]|[Hh][Ee][Xx])$`

`RtsCts`

Indica si se debe habilitar el control de flujo RTS/CTS. El valor predeterminado es `false`. Para obtener más información, consulte RTS, CTS, and RTR.

Nombre para mostrar en la consola AWS IoT: Control de flujo RTS/CTS

Obligatorio: `false`

Escriba: `string`

Valores válidos: `true, false`

Patrón válido: `^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`XonXoff`

Indica si se debe habilitar el control de flujo de software. El valor predeterminado es `false`. Para obtener más información, consulte Software flow control.

Nombre para mostrar en la consola AWS IoT: Control de flujo de software

Obligatorio: `false`

Escriba: `string`

Valores válidos: `true, false`

Patrón válido: `^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`Parity`

La paridad del puerto de serie. El valor predeterminado es `N`. Para obtener más información, consulte Parity.

Nombre para mostrar en la consola AWS IoT: Paridad de puerto serie

Obligatorio: `false`

Escriba: `string`

Valores válidos: `N, E, O, S, M`

Patrón válido: `^(|[NEOSMneosm])$`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector Serial Stream. Configura el conector para el modo de lectura de sondeo.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySerialStreamConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SerialStream/
versions/3",
            "Parameters": {
                "BaudRate" : "9600",
                "Timeout" : "25",
                "SerialPort" : "/dev/serial1",
                "SerialPort-ResourceId" : "my-serial-port-resource",
                "PollingRead" : "true",
                "PollingReadLength" : "30",
                "PollingReadInterval" : "30",
                "PollingReadType" : "hex"
            }
        }
    ]
```

```
}'
```

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta las solicitudes de lectura o escritura para los puertos de serie en dos temas de MQTT. Los mensajes de entrada deben tener un formato JSON válido.

- Solicitudes de lectura en el tema `serial/+/read/#`.
- Solicitudes de escritura en el tema `serial/+/write/#`.

Para publicar en estos temas, sustituya el comodín + con el nombre de cosa de núcleo y el comodín # con la ruta al puerto de serie. Por ejemplo:

```
serial/core-thing-name/read/dev/serial-port
```

Filtro de temas: `serial/+/read/#`

Consulte este tema para enviar las solicitudes de lectura bajo demanda a un pin de serie. Las solicitudes de lectura se limitan a una longitud de lectura máxima de 63994 bytes.

Propiedades de mensajes

`readLength`

La longitud de los datos para leer desde el puerto de serie.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[1-9][0-9]*$`

`type`

El tipo de datos para leer.

Obligatorio: `true`

Escriba: `string`

Valores válidos: `ascii, hex`

Patrón válido: `(?i)^(ascii|hex)$`

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a una respuesta de salida.

Obligatorio: `false`

Escriba: `string`

Patrón válido: `.+`

Ejemplo de entrada

```
{
    "readLength": "30",
    "type": "ascii",
    "id": "abc123"
}
```

Filtro de temas: `serial/+/write/#`

Use este tema para enviar solicitudes de escritura a un pin de serie. Las solicitudes de escritura se limitan a una longitud de datos máxima de 128000 bytes.

Propiedades de mensajes

`data`

La cadena para escribir en el puerto de serie.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[1-9][0-9]*$`

`type`

El tipo de datos para leer.

Obligatorio: `true`

Escriba: `string`

Valores válidos: `ascii, hex`

Patrón válido: `^(ascii|hex|ASCII|HEX)$`

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a una respuesta de salida.

Obligatorio: `false`

Escriba: `string`

Patrón válido: `.+`

Ejemplo entrada: solicitud ASCII

```
{
    "data": "random serial data",
    "type": "ascii",
    "id": "abc123"
}
```

Ejemplo entrada: solicitud hex

```
{
    "data": "base64 encoded data",
    "type": "hex",
    "id": "abc123"
}
```

Datos de salida

El conector publica los datos de salida en dos temas:

- La información del estado del conector en el tema `serial/+/status/#`.
- Las respuestas de solicitudes de lectura en el tema `serial/+/read_response/#`.

Al publicar en este tema, el conector sustituye el comodín + con el nombre de cosa de núcleo y el comodín # con la ruta al puerto de serie. Por ejemplo:

```
serial/core-thing-name/status/dev/serial-port
```

Filtro de temas: `serial/+/status/#`

Consulte este tema para escuchar el estado de las solicitudes de lectura y escritura. Si una propiedad `id` está incluida en la solicitud, se devuelve en la respuesta.

Ejemplo de salida: Correcto

```
{
    "response": {
        "status": "success"
    },
    "id": "abc123"
}
```

Ejemplo de salida: Error

Una respuesta de error incluye una propiedad `error_message` que describe el error o el tiempo de espera que se produce cuando se realiza la operación de lectura o escritura.

```
{
    "response": {
        "status": "fail",
        "error_message": "Could not write to port"
    },
    "id": "abc123"
}
```

Filtro de temas: `serial/+/read_response/#`

Utilice este tema para recibir los datos de respuesta de una operación de lectura. Los datos de respuesta son Base64 codificados si el tipo es `hex`.

Ejemplo de salida

```
{
    "data": "output of serial read operation"
```

```
    "id": "abc123"
}
```

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.
> - Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de Core AWS IoT Greengrass para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.
3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el recurso de dispositivo local requerido y conceda acceso de lectura/escritura a la función de Lambda.

   c. Agregue el conector a su grupo y configure sus parámetros.

   d. Agregue suscripciones al grupo que permitan que el conector reciba datos de entrada y envíe datos de salida en filtros de tema compatibles.

- Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
- Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.

4. Implemente el grupo.
5. En la consola de AWS IoT, en la página Prueba suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

   Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la característica envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector.

```
import greengrasssdk
import json

TOPIC_REQUEST = 'serial/CORE_THING_NAME/write/dev/serial1'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_serial_stream_request():
 request = {
  "data": "TEST",
  "type": "ascii",
  "id": "abc123"
 }
 return request

def publish_basic_request():
 iot_client.publish(payload=json.dumps(create_serial_stream_request()),
 topic=TOPIC_REQUEST)

publish_basic_request()
```

```
def lambda_handler(event, context):
 return
```

Licencias

El conector Serial Stream incluye el siguiente software/licencia de terceros:

- pyserial/BSD

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se actualizó el conector ARN para el soporte de Región de AWS.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”

Conector de integración MetricBase de ServiceNow

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

El conector de Integración ServiceNow MetricBase publica las métricas de serie temporal desde dispositivos Greengrass a ServiceNow MetricBase. Esto permite almacenar, analizar y visualizar datos de serie temporal desde el entorno del núcleo de Greengrass y actuar en eventos locales.

Este conector recibe datos de serie temporal en un tema de MQTT y los publica en la API de ServiceNow de forma periódica.

Puede utilizar este conector para admitir situaciones como:

- Crear alertas basadas en umbrales y alarmas en función de los datos de serie temporal recogidos mediante dispositivos Greengrass.
- Utilizar datos de servicios de tiempo desde dispositivos Greengrass con aplicaciones personalizadas integradas en la plataforma ServiceNow.

Este conector tiene las siguientes versiones.

Versión	ARN
4	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/4`
3	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/3`

Versión	ARN
2.	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/2`
1	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3 - 4

- El Software de AWS IoT Greengrass Core versión 1.9.3 o posterior AWS IoT Greengrass debe configurarse para admitir secretos locales, tal y como se describe en los requisitos de secretos.

  > **Note**
  >
  > Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

  > **Note**
  >
  > Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```

Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Una cuenta de ServiceNow con la suscripción activada en MetricBase. Además, se deben crear una métrica y una tabla de métricas en la cuenta. Para obtener más información, consulte MetricBase en la documentación de ServiceNow.
- Un secreto de tipo de texto en AWS Secrets Manager que almacena el nombre de usuario y la contraseña para iniciar sesión en la instancia de ServiceNow con la autenticación básica. El secreto debe contener las claves "user" and "password" con los valores correspondientes. Para obtener más información, consulte Creación de un secreto básico en la Guía del usuario de AWS Secrets Manager.
- Se crea un recurso de secretos en el grupo de Greengrass que hace referencia al secreto de Secrets Manager. Para obtener más información, consulte Implementación de secretos en el núcleo.

Versions 1 - 2

- AWS IoT Greengrass Software Core versión 1.7 o posterior. AWS IoT Greengrass debe configurarse para admitir secretos locales, como se describe en los Requisitos de secretos.

  Note

  Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Una cuenta de ServiceNow con la suscripción activada en MetricBase. Además, se deben crear una métrica y una tabla de métricas en la cuenta. Para obtener más información, consulte MetricBase en la documentación de ServiceNow.
- Un secreto de tipo de texto en AWS Secrets Manager que almacena el nombre de usuario y la contraseña para iniciar sesión en la instancia de ServiceNow con la autenticación básica. El secreto debe contener las claves "user" and "password" con los valores correspondientes. Para obtener más información, consulte Creación de un secreto básico en la Guía del usuario de AWS Secrets Manager.

- Se crea un recurso de secretos en el grupo de Greengrass que hace referencia al secreto de Secrets Manager. Para obtener más información, consulte Implementación de secretos en el núcleo.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

Version 4

`PublishInterval`

El número máximo de segundos que se debe esperar para la publicación de eventos en ServiceNow. El valor máximo es 900.

El conector se publica en ServiceNow cuando se alcanza `PublishBatchSize` o `PublishInterval` caduca.

Nombre para mostrar en la consola AWS IoT: Publicar intervalo en segundos

Obligatorio: `true`

Escriba: `string`

Valores válidos: `1 - 900`

Patrón válido: `[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

El número máximo de valores de métricas que se pueden procesar por lotes antes de que se publiquen en ServiceNow.

El conector se publica en ServiceNow cuando se alcanza `PublishBatchSize` o `PublishInterval` caduca.

Nombre para mostrar en la consola AWS IoT: Publicar tamaño del lote

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`InstanceName`

El nombre de la instancia que se utiliza para conectarse con ServiceNow.

Nombre para mostrar en la consola AWS IoT: Nombre de la instancia de ServiceNow

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`DefaultTableName`

El nombre de la tabla que contiene el parámetro `GlideRecord` asociado a la base de datos de MetricBase de serie temporal. La propiedad `table` de la carga de mensajes de entrada se puede utilizar para anular este valor.

Nombre para mostrar en la consola AWS IoT: Nombre de la tabla que contiene la métrica

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`MaxMetricsToRetain`

El número máximo de métricas para guardar en la memoria antes de que se sustituyen por nuevas métricas.

Este límite se aplica cuando no hay conexión a Internet y el conector comienza a almacenar en búfer las métricas que se van a publicar más adelante. Cuando el búfer está lleno, la métricas más antiguas se sustituyen por nuevas.

> **Note**
>
> Las métricas no se guardan si el proceso de host para el conector se interrumpe. Por ejemplo, este grupo puede ocurrir durante la implementación de grupos o cuando el dispositivo se reinicia.

Este valor debe ser mayor que el tamaño del lote y lo suficientemente grande como para almacenar mensajes en función de la velocidad de entrada de los mensajes de MQTT.

Nombre para mostrar en la consola AWS IoT: Máximo de métricas que se deben retener en la memoria

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`AuthSecretArn`

El secreto en AWS Secrets Manager que almacena la contraseña y el nombre de usuario de ServiceNow. Esto debe ser un secreto de tipo de texto. El secreto debe contener las claves "user" and "password" con los valores correspondientes.

Nombre para mostrar en la consola AWS IoT: ARN del secreto aut.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

El recurso de secretos del grupo que hace referencia al secreto del Secrets Manager para las credenciales de ServiceNow.

Nombre para mostrar en la consola AWS IoT: Recurso de token de autenticación

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`IsolationMode`

El modo de creación de contenedores para este conector. El valor predeterminado es `GreengrassContainer`, lo que significa que el conector se ejecuta en un entorno de tiempo de ejecución aislado dentro del contenedor de AWS IoT Greengrass.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo no se aplica a los conectores.

Nombre para mostrar en la consola AWS IoT: Modo de aislamiento de contenedores

Obligatorio: `false`

Escriba: `string`

Valores válidos: `GreengrassContainer` o `NoContainer`

Patrón válido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`PublishInterval`

El número máximo de segundos que se debe esperar para la publicación de eventos en ServiceNow. El valor máximo es 900.

El conector se publica en ServiceNow cuando se alcanza `PublishBatchSize` o `PublishInterval` caduca.

Nombre para mostrar en la consola AWS IoT: Publicar intervalo en segundos

Obligatorio: `true`

Escriba: `string`

Valores válidos: `1 - 900`

Patrón válido: `[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

El número máximo de valores de métricas que se pueden procesar por lotes antes de que se publiquen en ServiceNow.

El conector se publica en ServiceNow cuando se alcanza `PublishBatchSize` o `PublishInterval` caduca.

Nombre para mostrar en la consola AWS IoT: Publicar tamaño del lote

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`InstanceName`

El nombre de la instancia que se utiliza para conectarse con ServiceNow.

Nombre para mostrar en la consola AWS IoT: Nombre de la instancia de ServiceNow

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`DefaultTableName`

El nombre de la tabla que contiene el parámetro `GlideRecord` asociado a la base de datos de MetricBase de serie temporal. La propiedad `table` de la carga de mensajes de entrada se puede utilizar para anular este valor.

Nombre para mostrar en la consola AWS IoT: Nombre de la tabla que contiene la métrica

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`MaxMetricsToRetain`

El número máximo de métricas para guardar en la memoria antes de que se sustituyen por nuevas métricas.

Este límite se aplica cuando no hay conexión a Internet y el conector comienza a almacenar en búfer las métricas que se van a publicar más adelante. Cuando el búfer está lleno, la métricas más antiguas se sustituyen por nuevas.

> **Note**
>
> Las métricas no se guardan si el proceso de host para el conector se interrumpe. Por ejemplo, este grupo puede ocurrir durante la implementación de grupos o cuando el dispositivo se reinicia.

Este valor debe ser mayor que el tamaño del lote y lo suficientemente grande como para almacenar mensajes en función de la velocidad de entrada de los mensajes de MQTT.

Nombre para mostrar en la consola AWS IoT: Máximo de métricas que se deben retener en la memoria

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`AuthSecretArn`

El secreto en AWS Secrets Manager que almacena la contraseña y el nombre de usuario de ServiceNow. Esto debe ser un secreto de tipo de texto. El secreto debe contener las claves "user" and "password" con los valores correspondientes.

Nombre para mostrar en la consola AWS IoT: ARN del secreto aut.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

El recurso de secretos del grupo que hace referencia al secreto del Secrets Manager para las credenciales de ServiceNow.

Nombre para mostrar en la consola AWS IoT: Recurso de token de autenticación

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector de integración ServiceNow MetricBase.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyServiceNowMetricBaseIntegrationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ServiceNowMetricBaseIntegration/versions/4",
            "Parameters": {
                "PublishInterval" : "10",
                "PublishBatchSize" : "50",
                "InstanceName" : "myinstance",
                "DefaultTableName" : "u_greengrass_app",
                "MaxMetricsToRetain" : "20000",
                "AuthSecretArn" : "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "AuthSecretArn-ResourceId" : "MySecretResource",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de Conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta las métricas de serie temporal en un tema de MQTT y publica las métricas en ServiceNow. Los mensajes de entrada deben tener un formato JSON válido.

Filtro de temas en la suscripción

`servicenow/metricbase/metric`

Propiedades de mensajes

`request`

Información acerca de la tabla, el registro y la métrica. Esta solicitud representa el objeto `seriesRef` en una solicitud POST de serie temporal. Para obtener más información, consulte Clotho Time Series API - POST.

Obligatorio: `true`

Escriba: `object` que incluye las siguientes propiedades:

`subject`

El parámetro `sys_id` del registro específico en la tabla.

Obligatorio: `true`

Escriba: `string`

`metric_name`

El nombre del campo de métrica.

Obligatorio: `true`

Escriba: `string`

`table`

El nombre de la tabla en la que se almacenará el registro. Especifique este valor para anular el parámetro `DefaultTableName`.

Obligatorio: `false`

Escriba: `string`

`value`

El valor del punto de datos individual.

Obligatorio: `true`

Escriba: `float`

`timestamp`

La marca temporal del punto de datos individual. El valor predeterminado es la hora actual.

Obligatorio: `false`

Escriba: `string`

Ejemplo de entrada

```
{
    "request": {
        "subject":"ef43c6d40a0a0b5700c77f9bf387afe3",
        "metric_name":"u_count",
        "table": "u_greengrass_app"
        "value": 1.0,
        "timestamp": "2018-10-14T10:30:00"
    }
}
```

Datos de salida

Este conector publica información de estado como datos de salida en un tema MQTT.

Filtro de temas en la suscripción

`servicenow/metricbase/metric/status`

Ejemplo de salida: Correcto

```
{
    "response": {
        "metric_name": "Errors",
        "table_name": "GliderProd",
        "processed_on": "2018-10-14T10:35:00",
```

```
        "response_id": "khjKSkj132qwr23fcba",
        "status": "success",
        "values": [
            {
                "timestamp": "2016-10-14T10:30:00",
                "value": 1.0
            },
            {
                "timestamp": "2016-10-14T10:31:00",
                "value": 1.1
            }
        ]
    }
}
```

Ejemplo de salida: Error

```
{
    "response": {
        "error": "InvalidInputException",
        "error_message": "metric value is invalid",
        "status": "fail"
    }
}
```

> **Note**
>
> Si el conector detecta un error que se puede volver a intentar (por ejemplo, errores de conexión), volverá a intentar la publicación en el siguiente lote.

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.

- Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de AWS IoT Greengrass Core para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.
3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el recurso secreto requerido y conceda acceso de lectura a la función de Lambda.

   c. Agregue el conector y configure sus parámetros.

   d. Agregue suscripciones que permitan al conector recibir datos de entrada y enviar datos de salida en filtros de tema compatibles.

      - Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
      - Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.
4. Implemente el grupo.
5. En la consola de AWS IoT, en la página Prueba, suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la función envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
SEND_TOPIC = 'servicenow/metricbase/metric'

def create_request_with_all_fields():
    return {
        "request": {
             "subject": '2efdf6badbd523803acfae441b961961',
             "metric_name": 'u_count',
             "value": 1234,
             "timestamp": '2018-10-20T20:22:20',
             "table": 'u_greengrass_metricbase_test'
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=SEND_TOPIC,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licencias

El conector de integración ServiceNow MetricBase incluye las siguientes licencias y software de terceros:

- pysnow/MIT

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
4	Se ha agregado el parámetro `IsolationMode` para configurar el modo de creación de contenedores del conector.
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se ha introducido una corrección para reducir el registro excesivo.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”

Conector SNS

El conector SNS publica mensajes en un tema de Amazon SNS. Esto permite a los servidores web, direcciones de correo electrónico y otros suscriptores de mensajes responder a los eventos del grupo de Greengrass.

Este conector recibe información de mensaje de SNS sobre un tema de MQTT y, a continuación, envía el mensaje a un tema de SNS especificado. Si lo desea, puede utilizar las funciones de Lambda personalizadas para implementar los filtros o la lógica de formato en los mensajes antes de que se publiquen en este conector.

Este conector tiene las siguientes versiones.

Versión	ARN
4	`arn:aws:greengrass:region::/connectors/SNS/versions/4`
3	`arn:aws:greengrass:region::/connectors/SNS/versions/3`
2.	`arn:aws:greengrass:region::/connectors/SNS/versions/2`
1	`arn:aws:greengrass:region::/connectors/SNS/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3 - 4

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior.
- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

> **Note**
>
> Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Un tema de SNS configurado. Para obtener instrucciones, consulte el tema Creación de un tema de Amazon SNS en la Guía para desarrolladores de Amazon Simple Notification Service.
- El rol del grupo de Greengrass configurado para permitir la acción `sns:Publish` en el tema Amazon SNStopic de destino, tal y como se muestra en la siguiente política de IAM de ejemplo.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id:topic-name"
            ]
        }
    ]
}
```

Este conector le permite anular dinámicamente el tema predeterminado en la carga de mensajes de entrada. Si su implementación utiliza esta característica, la política de IAM debe permitir a `sns:Publish` el permiso en todos los temas de destino. Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *)

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Versions 1 - 2

- Software AWS IoT Greengrass Core versión 1.7 o posterior.
- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Un tema de SNS configurado. Para obtener instrucciones, consulte el tema Creación de un tema de Amazon SNS en la Guía para desarrolladores de Amazon Simple Notification Service.
- El rol del grupo de Greengrass configurado para permitir la acción `sns:Publish` en el tema Amazon SNStopic de destino, tal y como se muestra en la siguiente política de IAM de ejemplo.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id:topic-name"
            ]
        }
    ]
 }
```

Este conector le permite anular dinámicamente el tema predeterminado en la carga de mensajes de entrada. Si su implementación utiliza esta característica, la política de IAM debe permitir a `sns:Publish` el permiso en todos los temas de destino. Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *)

Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

Version 4

`DefaultSNSArn`

El ARN del tema de SNS predeterminado en el que publicar mensajes. El tema de destino se puede anular mediante la propiedad `sns_topic_arn` en la carga de mensajes de entrada.

> **Note**
>
> El rol de grupo debe permitir el permiso `sns:Publish` en todos los temas de destino. Para obtener más información, consulte the section called “Requisitos”.

Nombre para mostrar en la consola AWS IoT: ARN del tema SNS predeterminado

Obligatorio: `true`

Tipo: `string`

Patrón válido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`IsolationMode`

El modo de creación de contenedores para este conector. El valor predeterminado es `GreengrassContainer`, lo que significa que el conector se ejecuta en un entorno de tiempo de ejecución aislado dentro del contenedor de AWS IoT Greengrass.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo no se aplica a los conectores.

Nombre para mostrar en la consola AWS IoT: Modo de aislamiento de contenedores

Obligatorio: `false`

Tipo: `string`

Valores válidos: `GreengrassContainer` o `NoContainer`

Patrón válido: `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`DefaultSNSArn`

El ARN del tema de SNS predeterminado en el que publicar mensajes. El tema de destino se puede anular mediante la propiedad `sns_topic_arn` en la carga de mensajes de entrada.

> **Note**
>
> El rol de grupo debe permitir el permiso `sns:Publish` en todos los temas de destino. Para obtener más información, consulte the section called “Requisitos”.

Nombre para mostrar en la consola AWS IoT: ARN del tema SNS predeterminado

Obligatorio: `true`

Tipo: `string`

Patrón válido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector de SNS.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySNSConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SNS/versions/4",
            "Parameters": {
                "DefaultSNSArn": "arn:aws:sns:region:account-id:topic-name",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta la información de mensaje de SNS sobre un tema de MQTT y, a continuación, publica el mensaje tal cual en el tema de SNS de destino. Los mensajes de entrada deben tener un formato JSON válido.

Filtro de temas en la suscripción

`sns/message`

Propiedades de mensajes

`request`

Información sobre los mensajes que se envían al tema de SNS.

Obligatorio: `true`

Escriba: `object` que incluya las siguientes propiedades:

`message`

El contenido del mensaje como cadena o en formato JSON. Para ver ejemplos, consulte Ejemplo de entrada.

Para enviar JSON, la propiedad `message_structure` debe establecerse en `json` y el mensaje debe ser un objeto JSON codificado como cadena que contenga una clave `default`.

Obligatorio: `true`

Tipo: `string`

Patrón válido: `.*`

`subject`

El asunto del mensaje.

Obligatorio: `false`

Escriba: texto ASCII, de hasta 100 caracteres. Este debe empezar por una letra, un número o un signo de puntuación. Este no debe incluir saltos de línea ni caracteres de control.

Patrón válido: `.*`

`sns_topic_arn`

El ARN del tema de SNS en el que publicar mensajes. Si se especifica, el conector publica en este tema en lugar del tema predeterminado.

> **Note**
>
> El rol de grupo debe permitir el permiso `sns:Publish` en cualquier tema de destino. Para obtener más información, consulte the section called “Requisitos”.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`message_structure`

La estructura del mensaje.

Obligatorio: `false`. Se debe especificar para enviar un mensaje JSON.

Tipo: `string`

Valores válidos: `json`

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a una respuesta de salida. Si se especifica, la propiedad `id` en el objeto de respuesta se establece en este valor. Si no utiliza esta característica, puede omitir esta propiedad o especificar una cadena vacía.

Obligatorio: `false`

Tipo: `string`

Patrón válido: `.*`

Límites

El tamaño del mensaje está limitado a un tamaño de mensaje de SNS máximo de 256 KB.

Ejemplo de entrada: mensaje en cadena

Este ejemplo envía un mensaje en cadena. Especifica la propiedad `sns_topic_arn` opcional, que anula el tema de destino predeterminado.

```
{
    "request": {
        "subject": "Message subject",
        "message": "Message data",
        "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
    },
    "id": "request123"
}
```

Ejemplo de entrada: mensaje JSON

Este ejemplo envía un mensaje como objeto JSON codificado en cadena que incluye la clave `default`.

```
{
    "request": {
        "subject": "Message subject",
        "message": "{ \"default\": \"Message data\" }",
        "message_structure": "json"
    },
    "id": "request123"
}
```

Datos de salida

Este conector publica información de estado como datos de salida en un tema MQTT.

Filtro de temas en la suscripción

`sns/message/status`

Ejemplo de salida: Correcto

```
{
    "response": {
        "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
        "status": "success"
    },
    "id": "request123"
}
```

Ejemplo de salida: Error

```
{
   "response" : {
        "error": "InvalidInputException",
        "error_message": "SNS Topic Arn is invalid",
        "status": "fail"
   },
   "id": "request123"
```

```
}
```

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.
> - Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.

   Para el requisito de rol de grupo, debe configurar el rol para conceder los permisos necesarios y asegurarse de que el rol se ha añadido al grupo. Para obtener más información, consulte the section called “Administración del rol de grupo (consola)” o the section called “Administración del rol de grupo (CLI)”.

2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de AWS IoT Greengrass Core para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.

3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el conector y configure sus parámetros.

c. Agregue suscripciones que permitan al conector recibir datos de entrada y enviar datos de salida en filtros de tema compatibles.

   - Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
   - Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.

4. Implemente el grupo.
5. En la consola de AWS IoT, en la página Prueba suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

   Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la característica envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'sns/message'

def create_request_with_all_fields():
    return  {
        "request": {
            "message": "Message from SNS Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
```

```
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licencias

El conector de SNS incluye las siguientes licencias y software de terceros:

- AWS SDK for Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registros de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
4	Se ha agregado el parámetro `Isolation Mode` para configurar el modo de creación de contenedores del conector.
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se ha introducido una corrección para reducir el registro excesivo.

Versión	Cambios
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- Publicar acción en la documentación de Boto 3
- ¿Qué es Amazon Simple Notification Service? en la Guía para desarrolladores de Amazon Simple Notification Service

Conector de integración de Splunk

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

El conector de integración de Splunk publica los datos de los dispositivos Greengrass en Splunk. Esto permite usar Splunk para monitorizar y analizar el entorno del núcleo de Greengrass y actuar en eventos locales. El conector se integra con el recopilador de eventos HTTP (HEC). Para obtener más información, consulte Introduction to Splunk HTTP Event Collector en la documentación de Splunk.

Este conector recibe datos de eventos y registro en un tema de MQTT y publica los datos como están en la API de Splunk.

Puede utilizar este conector para admitir escenarios industriales como estos:

- Los operadores pueden utilizar datos periódicos de actuadores y sensores (por ejemplo, lecturas de temperatura, presión y agua) para desencadenar alarmas cuando se superen determinados umbrales.
- Los desarrolladores utilizar los datos recopilados de maquinaria industrial para crear modelos de machine learning que pueden monitorizar el equipamiento para buscar posibles problemas.

Este conector tiene las siguientes versiones.

Versión	ARN
4	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/4`
3	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/3`
2	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/2`
1	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 3 - 4

- El Software de AWS IoT Greengrass Core versión 1.9.3 o posterior AWS IoT Greengrass debe configurarse para admitir secretos locales, tal y como se describe en los requisitos de secretos

> **Note**
>
> Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

> **Note**
>
> Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- La funcionalidad del recopilador de eventos HTTP debe estar habilitado en Splunk. Para obtener más información, consulte Set up and use HTTP Event Collector in Splunk Web en la documentación de Splunk.
- Un secreto de tipo de texto en AWS Secrets Manager que almacena el token del recopilador de eventos HTTP. Para obtener más información, consulte Acerca de los tokens del recopilador de eventos en la documentación de Splunk y Creación de un secreto básico en la Guía del usuario de AWS Secrets Manager .

> **Note**
>
> Para crear el secreto en la consola de Secrets Manager, introduzca su token en la pestaña Plaintext. No incluya comillas ni ningún otro formato. En la API, especifique el token como valor de la propiedad `SecretString`.

- Se crea un recurso de secretos en el grupo de Greengrass que hace referencia al secreto de Secrets Manager. Para obtener más información, consulte Implementación de secretos en el núcleo.

Versions 1 - 2

- AWS IoT Greengrass Software Core versión 1.7 o posterior. AWS IoT Greengrass debe configurarse para admitir secretos locales, como se describe en los Requisitos de secretos.

  > **Note**
  >
  > Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- La funcionalidad del recopilador de eventos HTTP debe estar habilitado en Splunk. Para obtener más información, consulte Set up and use HTTP Event Collector in Splunk Web en la documentación de Splunk.
- Un secreto de tipo de texto en AWS Secrets Manager que almacena el token del recopilador de eventos HTTP. Para obtener más información, consulte Acerca de los tokens del recopilador de eventos en la documentación de Splunk y Creación de un secreto básico en la Guía del usuario de AWS Secrets Manager .

  > **Note**
  >
  > Para crear el secreto en la consola de Secrets Manager, introduzca su token en la pestaña Plaintext. No incluya comillas ni ningún otro formato. En la API, especifique el token como valor de la propiedad `SecretString`.

- Se crea un recurso de secretos en el grupo de Greengrass que hace referencia al secreto de Secrets Manager. Para obtener más información, consulte Implementación de secretos en el núcleo.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

Version 4

`SplunkEndpoint`

El punto de enlace de la instancia de Splunk. Este valor debe contener el protocolo, el nombre de host y el puerto.

Nombre para mostrar en la consola AWS IoT: Punto de conexión de Splunk

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

La cantidad de memoria (en KB) para asignar al conector.

Nombre que mostrar en la consola AWS IoT: Tamaño de memoria

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`SplunkQueueSize`

Número máximo de elementos que se van a guardar en la memoria antes de que los elementos se envíen o descarten. Cuando se alcanza este límite, se elementos más antiguos en la cola se reemplazan por los más recientes. Este límite normalmente se aplica cuando no existe conexión a Internet.

Nombre para mostrar en la consola AWS IoT: Número máximo de elementos que se deben retener

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`SplunkFlushIntervalSeconds`

El intervalo (en segundos) para publicar datos recibidos en HEC de Splunk. El valor máximo es 900. Para configurar el conector con el fin de publicar elementos a medida que se reciben (sin hacerlo por lotes), especifique 0.

Nombre para mostrar en la consola AWS IoT: Intervalo de publicación de Splunk

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

El secreto en AWS Secrets Manager que almacena el token de Splunk. Esto debe ser un secreto de tipo de texto.

Nombre para mostrar en la consola AWS IoT: ARN del secreto del token de autenticación de Splunk

Obligatorio: `true`

Escriba: `string`

Patrón válido: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

Se crea un recurso de secretos en el grupo de Greengrass que hace referencia al secreto de Splunk.

Nombre para mostrar en la consola AWS IoT: Recurso de token de autenticación de Splunk

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`SplunkCustomCALocation`

La ruta del archivo de la autoridad de certificación personalizada (CA) de Splunk (por ejemplo, `/etc/ssl/certs/splunk.crt`).

Nombre para mostrar en la consola AWS IoT: Ubicación de la autoridad de certificación personalizada de Splunk

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^$|/.*`

`IsolationMode`

El modo de creación de contenedores para este conector. El valor predeterminado es `GreengrassContainer`, lo que significa que el conector se ejecuta en un entorno de tiempo de ejecución aislado dentro del contenedor de AWS IoT Greengrass.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo no se aplica a los conectores.

Nombre para mostrar en la consola AWS IoT: Modo de aislamiento de contenedores

Obligatorio: `false`

Escriba: `string`

Valores válidos: `GreengrassContainer` o `NoContainer`

Patrón válido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`SplunkEndpoint`

El punto de enlace de la instancia de Splunk. Este valor debe contener el protocolo, el nombre de host y el puerto.

Nombre para mostrar en la consola AWS IoT: Punto de conexión de Splunk

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

La cantidad de memoria (en KB) para asignar al conector.

Nombre que mostrar en la consola AWS IoT: Tamaño de memoria

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`SplunkQueueSize`

Número máximo de elementos que se van a guardar en la memoria antes de que los elementos se envíen o descarten. Cuando se alcanza este límite, se elementos más antiguos en la cola se reemplazan por los más recientes. Este límite normalmente se aplica cuando no existe conexión a Internet.

Nombre para mostrar en la consola AWS IoT: Número máximo de elementos que se deben retener

Obligatorio: `true`

Escriba: `string`

Patrón válido: `^[0-9]+$`

`SplunkFlushIntervalSeconds`

El intervalo (en segundos) para publicar datos recibidos en HEC de Splunk. El valor máximo es 900. Para configurar el conector con el fin de publicar elementos a medida que se reciben (sin hacerlo por lotes), especifique 0.

Nombre para mostrar en la consola AWS IoT: Intervalo de publicación de Splunk

Obligatorio: `true`

Escriba: `string`

Patrón válido: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

El secreto en AWS Secrets Manager que almacena el token de Splunk. Esto debe ser un secreto de tipo de texto.

Nombre para mostrar en la consola AWS IoT: ARN del secreto del token de autenticación de Splunk

Obligatorio: `true`

Escriba: `string`

Patrón válido: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

Se crea un recurso de secretos en el grupo de Greengrass que hace referencia al secreto de Splunk.

Nombre para mostrar en la consola AWS IoT: Recurso de token de autenticación de Splunk

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`SplunkCustomCALocation`

La ruta del archivo de la autoridad de certificación personalizada (CA) de Splunk (por ejemplo, `/etc/ssl/certs/splunk.crt`).

Nombre para mostrar en la consola AWS IoT: Ubicación de la autoridad de certificación personalizada de Splunk

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^$|/.*`

Ejemplo de creación de conector (AWS CLI)

El siguiente comando de la CLI crea una `ConnectorDefinition` con una versión inicial que contiene el conector de integración de Splunk.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySplunkIntegrationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SplunkIntegration/
versions/4",
            "Parameters": {
                "SplunkEndpoint": "https://myinstance.cloud.splunk.com:8088",
                "MemorySize": 200000,
                "SplunkQueueSize": 10000,
                "SplunkFlushIntervalSeconds": 5,
                "SplunkTokenSecretArn":"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "SplunkTokenSecretArn-ResourceId": "MySplunkResource",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

> **Note**
>
> La función de Lambda de este conector tiene un ciclo de vida prolongado.

En la consola AWS IoT Greengrass, puede añadir un conector desde la página de conectores del grupo. Para obtener más información, consulte the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta datos de eventos y registro en un tema de MQTT y publica los datos recibidos como están en la API de Splunk. Los mensajes de entrada deben tener un formato JSON válido.

Filtro de temas en la suscripción

`splunk/logs/put`

Propiedades de mensajes

`request`

Los datos de eventos que se van a enviar a la API de Splunk. Los eventos debe cumplir las especificaciones de la API services/collector.

Obligatorio: `true`

Tipo:`object` Solo es obligatoria la propiedad `event`.

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a un estado de salida.

Obligatorio: `false`

Escriba: `string`

Límites

Todos los límites que son impuestos por la API de Splunk se aplican a las métricas cuando se utiliza este conector. Para obtener más información, consulte services/collector.

Ejemplo de entrada

```
{
    "request": {
        "event": "some event",
        "fields": {
            "severity": "INFO",
            "category": [
                "value1",
                "value2"
            ]
        }
    },
    "id": "request123"
}
```

Datos de salida

Este conector publica los datos de salida en dos temas:

- Información de estado en el tema `splunk/logs/put/status`.
- Los errores en el tema `splunk/logs/put/error`.

Filtro de temas: `splunk/logs/put/status`

Use este tema para escuchar el estado de las solicitudes. Cada vez que el conector envía un lote de los datos recibidos a la API de Splunk, publica una lista de los ID de las solicitudes que se han realizado correctamente y las solicitudes con errores.

Ejemplo de salida

```
{
    "response": {
        "succeeded": [
            "request123",
            ...
        ],
        "failed": [
            "request789",
            ...
        ]
    }
}
```

Filtro de temas: `splunk/logs/put/error`

Use este tema para escuchar errores del conector. La propiedad `error_message` que describe el error o el tiempo de espera que se producen cuando se espera al procesar la solicitud.

Ejemplo de salida

```
{
    "response": {
        "error": "UnauthorizedException",
        "error_message": "invalid splunk token",
        "status": "fail"
    }
```

```
}
```

> **Note**
>
> Si el conector detecta un error que se puede volver a intentar (por ejemplo, errores de conexión), volverá a intentar la publicación en el siguiente lote.

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> - Si usa otros tiempos de ejecución de Python, puede crear un enlace simbólico de Python3.x a Python 3.7.
> - Los temas Introducción a los conectores (consola) y Introducción a los conectores (CLI) contienen pasos detallados que muestran cómo configurar e implementar un conector de notificaciones Twilio de ejemplo.

1. Asegúrese de cumplir los requisitos para el conector.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de Core AWS IoT Greengrass para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.
3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).

   b. Agregue el recurso secreto requerido y conceda acceso de lectura a la función de Lambda.

c. Agregue el conector y configure sus parámetros.

d. Agregue suscripciones que permitan al conector recibir datos de entrada y enviar datos de salida en filtros de tema compatibles.

- Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
- Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.

4. Implemente el grupo.

5. En la consola de AWS IoT, en la página Prueba suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

   Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la característica envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'splunk/logs/put'

def create_request_with_all_fields():
    return {
        "request": {
            "event": "Access log test message."
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
```

```
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licencias

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
4	Se ha agregado el parámetro `Isolation Mode` para configurar el modo de creación de contenedores del conector.
3	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
2	Se ha introducido una corrección para reducir el registro excesivo.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”

- the section called “Introducción a los conectores (CLI)”

Conector de notificaciones Twilio

> **Warning**
>
> Este conector ha pasado a la fase de vida útil prolongada y AWS IoT Greengrass no lanzará actualizaciones que proporcionen funciones, mejoras de las funciones existentes, parches de seguridad o correcciones de errores. Para obtener más información, consulte AWS IoT Greengrass Version 1política de mantenimiento.

El conector de notificaciones Twilio realiza llamadas telefónicas automatizadas o envía mensajes de texto a través de Twilio. Puede utilizar este conector para enviar notificaciones en respuesta a los eventos en el grupo de Greengrass. Para llamadas telefónicas, el conector puede reenviar un mensaje de voz al destinatario.

Este conector recibe la información de mensaje de Twilio sobre un tema de MQTT y, a continuación, activa una notificación de Twilio.

> **Note**
>
> Para ver un tutorial que muestre cómo utilizar el conector de notificaciones Twilio, consulte the section called “Introducción a los conectores (consola)” o the section called “Introducción a los conectores (CLI)”.

Este conector tiene las siguientes versiones.

Versión	ARN
5	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/5`
4	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/4`

Versión	ARN
3	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/3`
2.	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/2`
1	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/1`

Para obtener información sobre los cambios de versión, consulte el Registro de cambios.

Requisitos

Este conector exige los siguientes requisitos:

Version 4 - 5

- Software AWS IoT Greengrass Core versión 1.9.3 o posterior. AWS IoT Greengrass debe configurarse para admitir secretos locales, como se describe en los Requisitos de secretos.

  > **Note**
  >
  > Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Python versión 3.7 o 3.8 instalado en el dispositivo principal y añadido a la variable de entorno PATH.

  > **Note**
  >
  > Para usar Python 3.8, ejecute el siguiente comando para crear un enlace simbólico desde la carpeta de instalación predeterminada de Python 3.7 a los binarios de Python 3.8 instalados.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass.

- Un SID de una cuenta de Twilio, un token de autenticación y un número de teléfono habilitado para Twilio. Después de crear un proyecto de Twilio, estos valores están disponibles en el panel del proyecto.

  > **Note**
  >
  > Puede utilizar una cuenta de prueba de Twilio. Si utiliza una cuenta de prueba, debe añadir números de teléfono de destinatarios que no sean de Twilio a una lista de números de teléfono verificados. Para obtener más información, consulte Cómo trabajar con su cuenta de prueba gratuita de Twilio.

- Un secreto de tipo de texto en AWS Secrets Manager que almacena el token de autenticación de Twilio. Para obtener más información, consulte Creación de un secreto básico en la Guía del usuario de AWS Secrets Manager.

  > **Note**
  >
  > Para crear el secreto en la consola de Secrets Manager, introduzca su token en la pestaña Plaintext. No incluya comillas ni ningún otro formato. En la API, especifique el token como valor de la propiedad `SecretString`.

- Se crea un recurso de secretos en el grupo de Greengrass que hace referencia al secreto de Secrets Manager. Para obtener más información, consulte Implementación de secretos en el núcleo.

Versions 1 - 3

- AWS IoT Greengrass Software Core versión 1.7 o posterior. AWS IoT Greengrass debe configurarse para admitir secretos locales, como se describe en los Requisitos de secretos.

> **Note**
>
> Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Versión 2.7 de Python instalada en el dispositivo principal y añadida a la variable de entorno PATH.
- Un SID de una cuenta de Twilio, un token de autenticación y un número de teléfono habilitado para Twilio. Después de crear un proyecto de Twilio, estos valores están disponibles en el panel del proyecto.

> **Note**
>
> Puede utilizar una cuenta de prueba de Twilio. Si utiliza una cuenta de prueba, debe añadir números de teléfono de destinatarios que no sean de Twilio a una lista de números de teléfono verificados. Para obtener más información, consulte Cómo trabajar con su cuenta de prueba gratuita de Twilio.

- Un secreto de tipo de texto en AWS Secrets Manager que almacena el token de autenticación de Twilio. Para obtener más información, consulte Creación de un secreto básico en la Guía del usuario de AWS Secrets Manager.

> **Note**
>
> Para crear el secreto en la consola de Secrets Manager, introduzca su token en la pestaña Plaintext. No incluya comillas ni ningún otro formato. En la API, especifique el token como valor de la propiedad `SecretString`.

- Se crea un recurso de secretos en el grupo de Greengrass que hace referencia al secreto de Secrets Manager. Para obtener más información, consulte Implementación de secretos en el núcleo.

Parámetros de conector

Este conector proporciona los siguientes parámetros:

Version 5

`TWILIO_ACCOUNT_SID`

El SID de cuenta de Twilio que se utiliza para invocar a la API de Twilio.

Nombre para mostrar en la consola AWS IoT: SID de la cuenta de Twilio

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`TwilioAuthTokenSecretArn`

El ARN del secreto del Secrets Manager que almacena el token de autenticación de Twilio.

> **Note**
>
> Este se usa para obtener acceso al valor del secreto local en el núcleo.

Nombre para mostrar en la consola AWS IoT: ARN del secreto del token de autenticación de Twilio

Obligatorio: `true`

Escriba: `string`

Patrón válido: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`TwilioAuthTokenSecretArn-ResourceId`

El ID del recurso de secreto del grupo de Greengrass que hace referencia al secreto del token de autenticación de Twilio.

Nombre para mostrar en la consola AWS IoT: Recurso del token de autenticación de Twilio

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`DefaultFromPhoneNumber`

El número de teléfono predeterminado habilitado para Twilio que Twilio utiliza para enviar mensajes. Twilio utiliza este número para iniciar el mensaje o la llamada.

- Si no configura un número de teléfono predeterminado, debe especificar un número de teléfono en la propiedad `from_number` en el cuerpo del mensaje de entrada.
- Si efectivamente configura un número de teléfono predeterminado, tiene la opción de anular el predeterminado especificando la propiedad `from_number` en el cuerpo del mensaje de entrada.

Nombre para mostrar en la consola AWS IoT: Predeterminado a partir del número de teléfono

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^$|\+[0-9]+`

`IsolationMode`

El modo de creación de contenedores para este conector. El valor predeterminado es `GreengrassContainer`, lo que significa que el conector se ejecuta en un entorno de tiempo de ejecución aislado dentro del contenedor de AWS IoT Greengrass.

> **Note**
>
> La configuración de creación de contenedores predeterminada para el grupo no se aplica a los conectores.

Nombre para mostrar en la consola AWS IoT: Modo de aislamiento en contenedores

Obligatorio: `false`

Escriba: `string`

Valores válidos: `GreengrassContainer` o `NoContainer`

Patrón válido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 4

`TWILIO_ACCOUNT_SID`

El SID de cuenta de Twilio que se utiliza para invocar a la API de Twilio.

Nombre para mostrar en la consola AWS IoT: SID de la cuenta de Twilio

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`TwilioAuthTokenSecretArn`

El ARN del secreto del Secrets Manager que almacena el token de autenticación de Twilio.

> **Note**
>
> Este se usa para obtener acceso al valor del secreto local en el núcleo.

Nombre para mostrar en la consola AWS IoT: ARN del secreto del token de autenticación de Twilio

Obligatorio: `true`

Escriba: `string`

Patrón válido: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`TwilioAuthTokenSecretArn-ResourceId`

El ID del recurso de secreto del grupo de Greengrass que hace referencia al secreto del token de autenticación de Twilio.

Nombre para mostrar en la consola AWS IoT: Recurso del token de autenticación de Twilio

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`DefaultFromPhoneNumber`

El número de teléfono predeterminado habilitado para Twilio que Twilio utiliza para enviar mensajes. Twilio utiliza este número para iniciar el mensaje o la llamada.

- Si no configura un número de teléfono predeterminado, debe especificar un número de teléfono en la propiedad `from_number` en el cuerpo del mensaje de entrada.
- Si efectivamente configura un número de teléfono predeterminado, tiene la opción de anular el predeterminado especificando la propiedad `from_number` en el cuerpo del mensaje de entrada.

Nombre para mostrar en la consola AWS IoT: Predeterminado a partir del número de teléfono

Obligatorio: `false`

Escriba: `string`

Patrón válido: `^$|\+[0-9]+`

Ejemplo de creación de conector (AWS CLI)

El siguiente ejemplo de comando de la CLI crea un parámetro `ConnectorDefinition` con una versión inicial que contiene el conector de notificaciones Twilio.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyTwilioNotificationsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/5",
            "Parameters": {
                "TWILIO_ACCOUNT_SID": "abcd12345xyz",
                "TwilioAuthTokenSecretArn": "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "TwilioAuthTokenSecretArn-ResourceId": "MyTwilioSecret",
                "DefaultFromPhoneNumber": "+19999999999",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
```

```
}'
```

Para ver tutoriales que muestran cómo añadir el conector de notificaciones de Twilio a un grupo, consulte the section called “Introducción a los conectores (CLI)” y the section called “Introducción a los conectores (consola)”.

Datos de entrada

Este conector acepta la información de mensaje de Twilio en dos temas de MQTT. Los mensajes de entrada deben tener un formato JSON válido.

- La información del mensaje de texto en el tema `twilio/txt`.
- La información del mensaje de teléfono en el tema `twilio/call`.

> **Note**
>
> La carga de mensajes de entrada puede incluir un mensaje de texto (`message`) o un mensaje de voz (`voice_message_location`), pero no ambos.

Filtro de temas: **`twilio/txt`**

Propiedades de mensajes

`request`

Información sobre la notificación de Twilio.

Obligatorio: `true`

Escriba: `object` que incluye las siguientes propiedades:

`recipient`

El destinatario del mensaje. Solo se admite un destinatario.

Obligatorio: `true`

Escriba: `object` que incluya las siguientes propiedades:

`name`

El nombre del destinatario.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.*`

`phone_number`

El número de teléfono del destinatario.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `\+[1-9]+`

`message`

El contenido de texto del mensaje de texto. Solo los mensajes de texto se admiten en este tema. Para mensajes de voz, utilice `twilio/call`.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`from_number`

El número de teléfono del remitente. Twilio utiliza este número de teléfono para iniciar el mensaje. Esta propiedad es necesaria si el parámetro `DefaultFromPhoneNumber` no está configurado. Si `DefaultFromPhoneNumber` está configurado, puede utilizar esta propiedad para anular el valor predeterminado.

Obligatorio: `false`

Escriba: `string`

Patrón válido: `\+[1-9]+`

`retries`

El número de reintentos. El valor predeterminado es 0.

Obligatorio: `false`

Escriba: `integer`

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a una respuesta de salida.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

Ejemplo de entrada

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "message": "Hello from the edge"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

Filtro de temas: **`twilio/call`**

Propiedades de mensajes

`request`

Información sobre la notificación de Twilio.

Obligatorio: `true`

Escriba: `object` que incluya las siguientes propiedades:

`recipient`

El destinatario del mensaje. Solo se admite un destinatario.

Obligatorio: `true`

Escriba: `object` que incluya las siguientes propiedades:

`name`

El nombre del destinatario.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`phone_number`

El número de teléfono del destinatario.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `\+[1-9]+`

`voice_message_location`

La URL del contenido de audio para el mensaje de voz. Esta debe estar en formato TwiML. Solo los mensajes de voz se admiten en este tema. Para mensajes de texto, utilice `twilio/txt`.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

`from_number`

El número de teléfono del remitente. Twilio utiliza este número de teléfono para iniciar el mensaje. Esta propiedad es necesaria si el parámetro `DefaultFromPhoneNumber` no está configurado. Si `DefaultFromPhoneNumber` está configurado, puede utilizar esta propiedad para anular el valor predeterminado.

Obligatorio: `false`

Escriba: `string`

Patrón válido: `\+[1-9]+`

`retries`

El número de reintentos. El valor predeterminado es 0.

Obligatorio: `false`

Escriba: `integer`

`id`

Un ID arbitrario para la solicitud. Esta propiedad se usa para asignar una solicitud de entrada a una respuesta de salida.

Obligatorio: `true`

Escriba: `string`

Patrón válido: `.+`

Ejemplo de entrada

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "voice_message_location": "https://some-public-TwiML"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

Datos de salida

Este conector publica información de estado como datos de salida en un tema MQTT.

Filtro de temas en la suscripción

`twilio/message/status`

Ejemplo de salida: Correcto

```
{
```

```
    "response": {
        "status": "success",
        "payload": {
            "from_number": "+19999999999",
            "messages": {
                "message_status": "queued",
                "to_number": "+12345000000",
                "name": "Darla"
            }
        }
    },
    "id": "request123"
}
```

Ejemplo de salida: Error

```
{
    "response": {
        "status": "fail",
        "error_message": "Recipient name cannot be None",
        "error": "InvalidParameter",
        "payload": None
        }
    },
    "id": "request123"
}
```

La propiedad `payload` en la salida es la respuesta de la API de Twilio al enviar el mensaje. Si el conector detecta que los datos de entrada no son válidos (por ejemplo, no especifica un campo de entrada requerido), el conector devuelve un error y establece el valor en `None`. A continuación se muestran cargas de ejemplo:

```
{
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'undelivered'
    }
}
```

```
{
```

```
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'queued'
    }
}
```

Ejemplo de uso

Utilice los siguientes pasos de alto nivel para configurar una función de Lambda de Python 3.7 de ejemplo que puede utilizar para probar el conector.

> **Note**
>
> Los temas the section called “Introducción a los conectores (consola)” y the section called “Introducción a los conectores (CLI)” contienen pasos de extremo a extremo que muestran cómo configurar, implementar y probar el conector de notificaciones de Twilio.

1. Asegúrese de cumplir los requisitos para el conector.
2. Cree y publique una función de Lambda que envíe datos de entrada al conector.

   Guarde el código de ejemplo como un archivo PY. Descargue y descomprima el SDK de AWS IoT Greengrass Core para Python. A continuación, cree un paquete zip que contenga el archivo PY y la carpeta `greengrasssdk` en el nivel raíz. Este paquete zip es el paquete de implementación que se carga en AWS Lambda.

   Después de crear la función de Lambda de Python 3.7, publique una versión de característica y cree un alias.
3. Configuración del grupo de Greengrass.

   a. Agregue la función de Lambda por su alias (recomendado). Configure el ciclo de vida de Lambda como de larga duración (o `"Pinned": true` en la CLI).
   b. Agregue el recurso secreto requerido y conceda acceso de lectura a la función de Lambda.
   c. Agregue el conector y configure sus parámetros.
   d. Agregue suscripciones que permitan al conector recibir datos de entrada y enviar datos de salida en filtros de tema compatibles.

- Establezca la función de Lambda como fuente, el conector como destino y utilice un filtro de tema de entrada compatible.
- Establezca el conector como origen, AWS IoT Core como destino y utilice un filtro de tema de salida compatible. Utilice esta suscripción para ver los mensajes de estado en la consola de AWS IoT.

4. Implemente el grupo.
5. En la consola de AWS IoT, en la página Prueba suscríbase al tema de datos de salida para ver los mensajes de estado del conector. La función de Lambda de ejemplo es de larga duración y comienza a enviar mensajes inmediatamente después de implementar el grupo.

   Cuando haya terminado de probar, puede establecer el ciclo de vida de Lambda en Bajo demanda (o `"Pinned": false` en la CLI) e implementar el grupo. Esto impide que la función envíe mensajes.

Ejemplo

El siguiente ejemplo de función de Lambda envía un mensaje de entrada al conector. En este ejemplo, se activa un mensaje de texto.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
TXT_INPUT_TOPIC = 'twilio/txt'
CALL_INPUT_TOPIC = 'twilio/call'

def publish_basic_message():

    txt = {
        "request": {
            "recipient" : {
                "name": "Darla",
                "phone_number": "+12345000000",
                "message": 'Hello from the edge'
            },
            "from_number" : "+19999999999"
        },
        "id" : "request123"
    }
```

```
    print("Message To Publish: ", txt)

    client.publish(topic=TXT_INPUT_TOPIC,
                   payload=json.dumps(txt))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licencias

El conector de notificaciones Twilio incluye las siguientes licencias y software de terceros:

- twilio-python/MIT

Este conector se publica bajo el contrato de licencia de software de Greengrass Core.

Registro de cambios

La siguiente tabla describe los cambios en cada versión del conector.

Versión	Cambios
5	Se ha agregado el parámetro `Isolation Mode` para configurar el modo de creación de contenedores del conector.
4	Se actualizó el tiempo de ejecución de Lambda a Python 3.7, lo que cambia el requisito de tiempo de ejecución.
3	Se ha introducido una corrección para reducir el registro excesivo.
2	Pequeñas correcciones de errores y mejoras.
1	Versión inicial.

Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “Introducción a los conectores (consola)”
- the section called “Introducción a los conectores (CLI)”
- Twilio API Reference

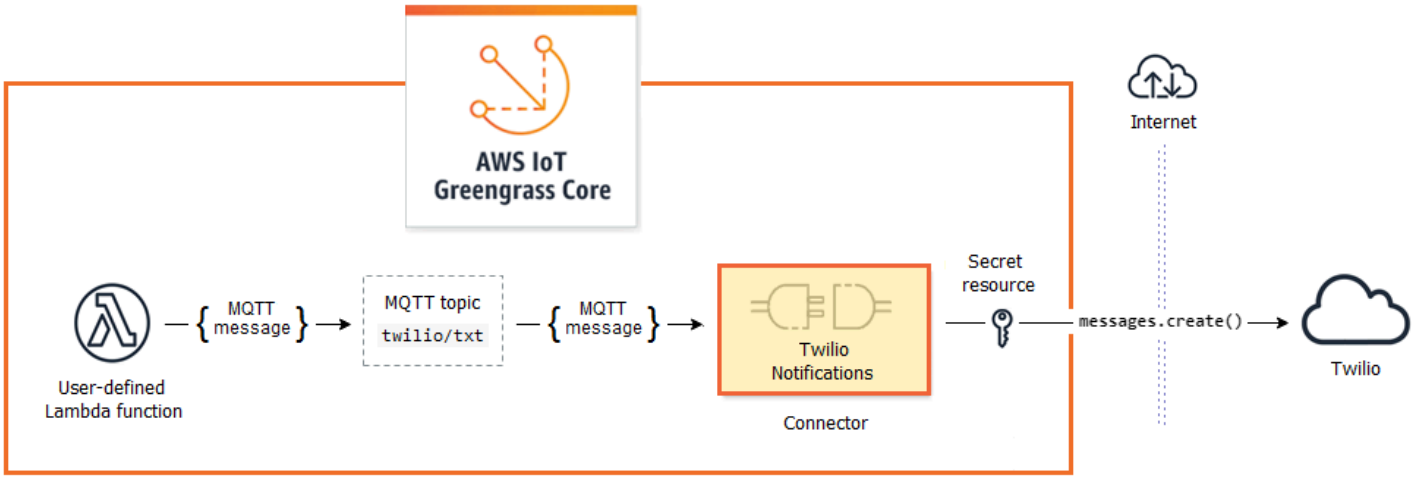
Introducción a los conectores de Greengrass (consola)

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

Este tutorial muestra cómo utilizar la AWS Management Console para trabajar con conectores.

Utilice conectores para acelerar el ciclo de vida de desarrollo. Los conectores son módulos precompilados y reutilizables que puede simplificar la interacción con los servicios, los protocolos y los recursos. Pueden ayudarle a implementar la lógica de negocio en los dispositivos de Greengrass con mayor rapidez. Para obtener más información, consulte Integración con servicios y protocolos mediante conectores.

En este tutorial configurará e implementará el conector de Notificaciones de Twilio. El conector recibe la información del mensaje de Twilio como datos de entrada y, a continuación, desencadena un mensaje de texto de Twilio. El flujo de datos se muestra en el siguiente diagrama.



Después de configurar el conector, debe crear una función de Lambda y una suscripción.

- La función evalúa los datos simulados desde un sensor de temperatura. Publica condicionalmente la información del mensaje de Twilio a un tema de MQTT. Este es el tema al que se suscribe el conector.
- La suscripción permite que la función se publique en el tema y que el conector reciba datos del tema.

El conector de Notificaciones de Twilio necesita un token de autenticación de Twilio para interactuar con la API de Twilio. El token es un secreto de tipo de texto creado en AWS Secrets Manager y al que se hace referencia desde un recurso de grupo. Esto permite a AWS IoT Greengrass crear una copia local del secreto en el núcleo de Greengrass, donde se cifra y se convierte en disponible para el conector. Para obtener más información, consulte Implementación de secretos en el núcleo.

El tutorial contiene los siguientes pasos generales:

1. Crear un secreto en Secrets Manager
2. Agregar un recurso de secreto a un grupo
3. Agregar un conector al grupo
4. Creación de un paquete de implementación de la función de Lambda
5. Creación de una funcion de Lambda
6. Agregar una función al grupo
7. Agregar suscripciones al grupo

8. Implementar el grupo
9. the section called “Probar la solución”

Completar el tutorial debería tomarle aproximadamente 20 minutos.

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Un grupo de Greengrass y un núcleo de Greengrass (v1.9.3 o posterior). Para obtener información acerca de cómo crear un núcleo y un grupo de Greengrass, consulte Empezando con AWS IoT Greengrass. El tutorial de introducción también incluye pasos para instalar el software AWS IoT Greengrass Core.
- Python 3.7 instalado en el dispositivo central de AWS IoT Greengrass.
- AWS IoT Greengrass debe configurarse para admitir secretos locales, como se describe en Requisitos de secretos.

  > **Note**
  >
  > Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Un SID de una cuenta de Twilio, un token de autenticación y un número de teléfono habilitado para Twilio. Después de crear un proyecto de Twilio, estos valores están disponibles en el panel del proyecto.

  > **Note**
  >
  > Puede utilizar una cuenta de prueba de Twilio. Si utiliza una cuenta de prueba, debe añadir números de teléfono de destinatarios que no sean de Twilio a una lista de números de teléfono verificados. Para obtener más información, consulte Cómo trabajar con su cuenta de prueba gratuita de Twilio.

Paso 1: Crear un secreto de Secrets Manager

En este paso, utilice la consola AWS Secrets Manager para crear un secreto de tipo de texto para su token de autenticación de Twilio.

1. Inicie sesión en la consola de AWS Secrets Manager.

   > **Note**
   >
   > Para obtener más información sobre este proceso, consulte Paso 1: Crear y almacenar el secreto en AWS Secrets Manager en la Guía del usuario de AWS Secrets Manager.

2. Elija Store a new secret (Almacenar un nuevo secreto).
3. En Seleccionar tipo de secreto, elija Otro tipo de secretos.
4. En Specify the key/value pairs to be stored for this secret (Especificar los pares clave-valor que se almacenarán para este secreto), en la pestaña Plaintext (Texto no cifrado), escriba su token de autenticación de Twilio. Quite todo el formato JSON e introduzca solo el valor del token.
5. Mantenga aws/secretsmanager seleccionado para la clave de cifrado y, a continuación, seleccione Siguiente.

   > **Note**
   >
   > AWS KMS no le cobrará si utiliza la clave administrada de AWS predeterminada que Secrets Manager crea en su cuenta.

6. En Secret name (Nombre del secreto), escriba **`greengrass-TwilioAuthToken`** y, a continuación, seleccione Next (Siguiente).

   > **Note**
   >
   > De forma predeterminada, el rol de servicio Greengrass permite a AWS IoT Greengrass obtener el valor de los secretos con nombres que comienzan por greengrass-. Para obtener más información, consulte requisitos de secretos.

7. Este tutorial no requiere rotación, así que elija desactivar la rotación automática y, a continuación, seleccione Siguiente.
8. En la página Review (Revisar), revise los ajustes y, a continuación, seleccione Store (Almacenar).

A continuación, se crea un recurso de secreto en su grupo de Greengrass que hace referencia al secreto.

Paso 2: Agregar un recurso de secreto a un grupo de Greengrass

En este paso, va a añadir un recurso de secreto al grupo de Greengrass. Este recurso es una referencia al secreto que creó en el paso anterior.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija el grupo al que desee añadir el recurso de secreto.
3. En la página de configuración del grupo, elija la pestaña Recursos y, a continuación, desplácese hacia abajo hasta la sección Secretos. La sección Secretos muestra los recursos secretos que pertenecen al grupo. Puede añadir, editar y quitar los recursos de secreto de esta sección.

   > **Note**
   >
   > Como alternativa, la consola le permite crear un secreto y un recurso secreto al configurar un conector o una función de Lambda. Puede hacerlo desde la página Configurar parámetros del conector o desde la página Recursos de la función de Lambda.

4. Seleccione Añadir en la sección Secretos.
5. En la página Añadir un recurso secreto, introduzca **`MyTwilioAuthToken`** el nombre del recurso.
6. Para el secreto, elija Greengrass-TwilioAuthToken.
7. En la sección Seleccionar etiquetas (opcional), la etiqueta provisional AWSCURRENT representa la versión más reciente del secreto. Esta etiqueta siempre está incluida en un recurso de secreto.

   > **Note**
   >
   > Este tutorial solo requiere la etiqueta AWSCURRENT. De forma opcional, puede incluir etiquetas que requieran su función de Lambda o un conector.

8. Seleccione Add resource (Añadir recurso).

Paso 3: Agregar un conector al grupo de Greengrass

En este paso va a configurar parámetros para el conector Notificaciones de Twilio y añadirlo al grupo.

1. En la página de configuración de grupo, elija Connectors (Conectores) y, a continuación, elija Add a connector (Añadir un conector).
2. En la página Añadir conector, selecciona Notificaciones de Twilio.
3. Elija la versión .
4. En la sección Configuración:
   - Para el recurso de token de autenticación de Twilio, introduzca el recurso que creó en el paso anterior.

     > **Note**
     >
     > Cuando se introduce el recurso, la propiedad ARN de secreto de token de autenticación de Twilio se rellena automáticamente.

   - En Default from phone number (Predeterminado de número de teléfono), introduzca su número de teléfono habilitado para Twilio.
   - Para Twilio account SID (SID de cuenta de Twilio), introduzca su SID de cuenta de Twilio.
5. Seleccione Add resource (Añadir recurso).

Paso 4: Crear un paquete de implementación de la función de Lambda

Para crear una función de Lambda, primero debe crear un paquete de implementación de funciones de Lambda que contenga el código de la función y las dependencias. Las funciones de Lambda de Greengrass requieren el SDK de AWS IoT Greengrass Core para tareas como la comunicación con los mensajes de MQTT en el entorno principal y el acceso a los secretos locales. En este tutorial se crea una característica de Python para que utilices la versión Python del SDK en el paquete de implementación.

1. Desde la página de descargas del Core SDK AWS IoT Greengrass, descargue el SDK AWS IoT Greengrass básico para Python en su ordenador.
2. Descomprima el paquete descargado para obtener el SDK. El SDK es la carpeta `greengrasssdk`.

3. Guarde la siguiente función de código de Python en un archivo local llamado "`temp_monitor.py`".

```
import greengrasssdk
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. Comprima en un archivo ZIP los siguientes elementos en un archivo denominado "`temp_monitor_python.zip`". Al crear el archivo ZIP, incluya únicamente el código y sus dependencias, no la carpeta donde se encuentran.

- temp_monitor.py. Lógica de la aplicación.
- greengrasssdk. Biblioteca necesaria para las funciones de Lambda de Python Greengrass que publican mensajes MQTT.

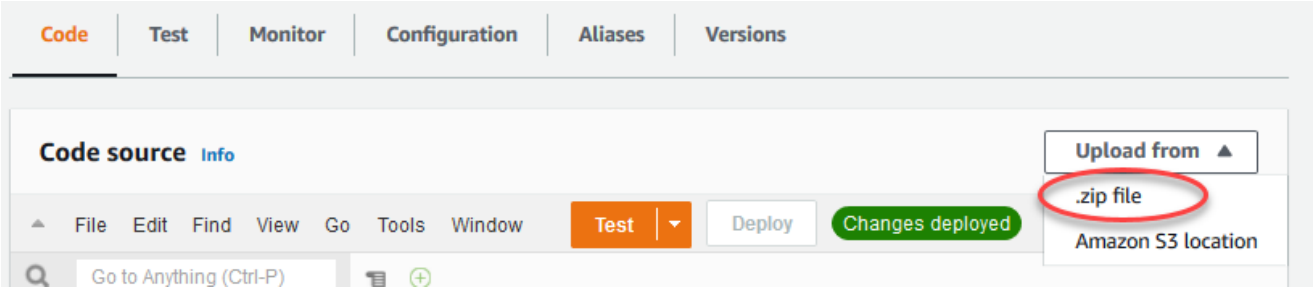
Este es el paquete de implementación de la función de Lambda.

Ahora, cree una función de Lambda que use el paquete de implementación.

Paso 5: Crear una segunda función de Lambda en la consola de AWS Lambda

En este paso, va a utilizar la consola de AWS Lambda para crear una función de Lambda y va a configurarla para utilizar el paquete de implementación. A continuación, publicará una versión de la característica y creará un alias.

1. Primero, cree una función de Lambda.
   a. En la AWS Management Console, elija Servicios y abra la consola de AWS Lambda.
   b. Elija Crear función, y, a continuación, elija Autor desde cero.
   c. En la sección Basic information (Información básica), utilice los siguientes valores:
      - En Function name (Nombre de la función), introduzca **`TempMonitor`**.
      - En Runtime (Tiempo de ejecución), elija Python 3.7.
      - En Permisos, mantenga la configuración predeterminada. Esto crea un rol de ejecución que otorga permisos Lambda básicos. AWS IoT Greengrass no utiliza este rol.
   d. En la parte inferior de la página, elija Create function.
2. A continuación, registre el controlador y cargue el paquete de implementación de la función de Lambda.
   a. En la pestaña Código, en Código fuente, seleccione Cargar desde. En el menú desplegable, seleccione un archivo .zip..



b. Seleccione Cargar y, a continuación, elija su paquete de implementación `temp_monitor_python.zip`. A continuación, elija Guardar.

c. En la pestaña Código de la función, en Configuración de tiempo de ejecución, elija Editar y, a continuación, introduzca los siguientes valores.

- En Runtime (Tiempo de ejecución), elija Python 3.7.
- En Handler (Controlador), escriba **`temp_monitor.function_handler`**.

d. Seleccione Save.

> **Note**
>
> El botón de prueba de la consola de AWS Lambda no funciona con esta función. El SDK AWS IoT Greengrass Core no contiene los módulos necesarios para ejecutar las funciones de Lambda de Greengrass de forma independiente en la consola AWS Lambda. Estos módulos (por ejemplo, `greengrass_common`) se suministran a las funciones una vez desplegados en el núcleo de Greengrass.

3. Ahora, publique la primera versión de su función de Lambda y cree un alias para la versión.

> **Note**
>
> Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

a. En el menú Actions, elija Publish new version.

b. En Version description (Descripción de versión), escriba **`First version`** y, a continuación, elija Publish (Publicar).

c. En la página de configuración de TempMonitor: 1, en el menú Actions (Acciones), elija Create alias (Crear alias).

d. En la página Create a new alias, utilice los valores siguientes:

- En Name (Nombre), ingrese **`GG_TempMonitor`**.
- En Version (Versión), elija 1.

> **Note**
>
> AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST.

e. Seleccione Create (Crear).

Ahora está preparado para añadir la función de Lambda al grupo de Greengrass.

Paso 6: Agregar una función de Lambda al grupo de Greengrass

En este paso, va a añadir la función de Lambda al grupo y a configurar el ciclo de vida y las variables de entorno. Para obtener más información, consulte the section called “Control de la ejecución de la función de Lambda de Greengrass”.

1. En la página de configuración del grupo, elija la pestaña Funciones de Lambda.
2. En la sección Mis funciones de Lambda, seleccione Añadir.
3. En la página Agregar función de Lambda, elija TempMonitor para la función de Lambda.
4. Para la versión de la función de Lambda, elija Alias: GG_TempMonitor.
5. Elija Añadir función de Lambda.

Paso 7: Agregar suscripciones al grupo de Greengrass

En este paso añadirá una suscripción que habilita la función de Lambda para enviar datos de entrada al conector. El conector define los temas de MQTT a los que se suscribe, por lo que esta suscripción utiliza uno de los temas. Este es el mismo tema en el que se publica la función de ejemplo.

En este tutorial también creará suscripciones que permitan a la función recibir lecturas de temperatura simuladas de AWS IoT y permitan a AWS IoT recibir información sobre el estado del conector.

1. En la página de configuración del grupo, elija la pestaña Suscripciones y, a continuación, elija Añadir suscripción.
2. En la página Crear una suscripción, configure el origen y el destino de la siguiente manera:
   a. Para Tipo de origen, elija Función de lambda y, a continuación, TempMonitor.
   b. Para Tipo de destino, seleccione Conector y, a continuación, seleccione Notificaciones de Twilio.
3. En el Filtro de temas, escriba **`twilio/txt`**.
4. Elija Crear una suscripción.
5. Repita los pasos del 1 al 4 para crear una suscripción que permita a AWS IoT publicar mensajes en la función.
   a. Para Tipo de origen, elija Servicio y, a continuación, Nube de IoT.
   b. En Seleccionar un destino, elija Funciones de Lambda y, a continuación, elija TempMonitor.
   c. En Topic filter (Filtro de temas), escriba **`temperature/input`**.
6. Repita los pasos del 1 al 4 para crear una suscripción que permita al conector publicar mensajes en AWS IoT.
   a. Para Tipo de origen, seleccione Conector y, a continuación, seleccione Notificaciones de Twilio.
   b. Para Tipo de destino, elija Servicio y, a continuación, Nube de IoT.
   c. En Topic filter (Filtro de temas), se introduce **`twilio/message/status`** automáticamente. Este es el tema predefinido en el que publica el conector.

Paso 8: Implementar el grupo de Greengrass

Implemente el grupo en el dispositivo del núcleo.

1. Asegúrese de que el núcleo de AWS IoT Greengrass se está ejecutando. Ejecute los siguientes comandos en el terminal de Raspberry Pi según sea necesario.
   a. Para comprobar si el daemon está en ejecución:

```
ps aux | grep -E 'greengrass.*daemon'
```

Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, el daemon está en ejecución.

> **Note**
>
> La versión que figura en la ruta depende de la versión del software AWS IoT Greengrass Core que esté instalada en el dispositivo del núcleo.

b. Iniciar el daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. En la página de configuración de grupo, elija Implementar.
3. a. En la pestaña Funciones de Lambda, en la sección Funciones de Lambda del sistema, seleccione Detector IP y elija Editar.

   b. En el cuadro de diálogo Editar configuración del detector IP, seleccione Detectar y anular automáticamente los puntos de conexión del agente MQTT.

   c. Seleccione Save.

   Esto permite a los dispositivos adquirir automáticamente la información de conexión del dispositivo principal, como la dirección IP, el DNS y el número de puerto. Se recomienda la detección automática, pero AWS IoT Greengrass también es compatible con puntos de conexión especificados manualmente. Solo se le solicitará el método de detección la primera vez que se implemente el grupo.

   > **Note**
   >
   > Si se le solicita, conceda permiso para crear el rol de servicio de Greengrass y asócielo a su Cuenta de AWS en el Región de AWS actual. Este rol permite a AWS IoT Greengrass acceder a los servicios de AWS.

En la página Deployments (Implementaciones), se muestra la marca temporal, el ID de versión y el estado de la implementación. Una vez terminada, la implementación debería mostrar el estado Completado.

Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

> **Note**
>
> Un grupo de Greengrass solo puede contener una versión del conector a la vez. Para obtener información sobre cómo actualizar una versión de conector, consulte the section called “Actualización de versiones de los conectores”.

Probar la solución

1. En la página de inicio de la consola AWS IoT, elija Pruebas.
2. Para Suscribirse al tema, utilice los siguientes valores y, a continuación, seleccione Suscribirse. El conector Notificaciones de Twilio publica información sobre el estado en este tema.

Propiedad	Valor
Subscription topic	twilio/message/status
Visualización de la carga de MQTT	Display payloads as strings

3. Para Publicar en tema, utilice los siguientes valores y, a continuación, seleccione Publicar para invocar la función.

Propiedad	Valor
Tema	temperature/input
Mensaje	Sustituya *recipient-name* con un nombre y *recipient-phone-number* con el número de teléfono del destinatario

Propiedad	Valor
	del mensaje de texto. Ejemplo: `+12345000000` `{ "to_name": " recipient-name ", "to_number": " recipient-phone-number ", "temperature": 31 }` Si utiliza una cuenta de prueba, debe añadir números de teléfono de destinatarios que no sean de Twilio a una lista de números de teléfono verificados. Para obtener más información, consulte Verificar su número de teléfono personal.

Si se ejecuta correctamente, el destinatario recibe el mensaje de texto y la consola muestra el estado `success` de los datos de salida.

Ahora, cambie el valor de `temperature` en el mensaje de entrada a **29** y publíquelo. Dado que es menos de 30, la función TempMonitor no activa un mensaje de Twilio.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “conectores de Greengrass proporcionados por AWS”

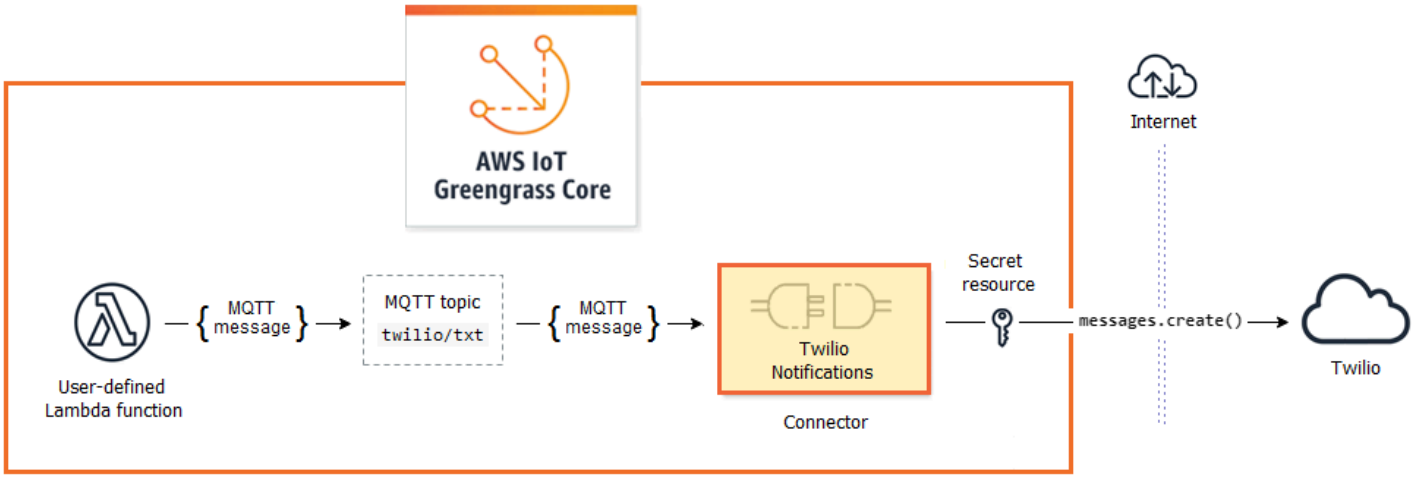
Introducción a los conectores de Greengrass (CLI)

Esta característica está disponible para AWS IoT Greengrass Core versión 1.7 y posteriores.

Este tutorial muestra cómo utilizar la AWS CLI para trabajar con conectores.

Utilice conectores para acelerar el ciclo de vida de desarrollo. Los conectores son módulos precompilados y reutilizables que puede simplificar la interacción con los servicios, los protocolos y los recursos. Pueden ayudarle a implementar la lógica de negocio en los dispositivos de Greengrass con mayor rapidez. Para obtener más información, consulte Integración con servicios y protocolos mediante conectores.

En este tutorial configurará e implementará el conector de Notificaciones de Twilio. El conector recibe la información del mensaje de Twilio como datos de entrada y, a continuación, desencadena un mensaje de texto de Twilio. El flujo de datos se muestra en el siguiente diagrama.



Después de configurar el conector, debe crear una función de Lambda y una suscripción.

- La función evalúa los datos simulados desde un sensor de temperatura. Publica condicionalmente la información del mensaje de Twilio a un tema de MQTT. Este es el tema al que se suscribe el conector.
- La suscripción permite que la función se publique en el tema y que el conector reciba datos del tema.

El conector de Notificaciones de Twilio necesita un token de autenticación de Twilio para interactuar con la API de Twilio. El token es un secreto de tipo de texto creado en AWS Secrets Manager y al que se hace referencia desde un recurso de grupo. Esto permite a AWS IoT Greengrass crear una copia local del secreto en el núcleo de Greengrass, donde se cifra y se convierte en disponible para el conector. Para obtener más información, consulte Implementación de secretos en el núcleo.

El tutorial contiene los siguientes pasos generales:

1. Crear un secreto en Secrets Manager
2. Crear una versión y una definición del recurso
3. Crear una versión y una definición del conector
4. Creación de un paquete de implementación de la función de Lambda
5. Creación de una función de Lambda
6. Crear una versión y una definición de la función
7. Crear una versión y una definición de la suscripción
8. Cree una versión del grupo
9. Crear una implementación
10. the section called “Probar la solución”

Completar el tutorial debería tomarle aproximadamente 30 minutos.

Uso de la API de AWS IoT Greengrass

Es útil para comprender los siguientes patrones a la hora de trabajar con componentes de grupo y grupos de Greengrass (por ejemplo, los conectores, las funciones y los recursos del grupo).

- En la parte superior de la jerarquía, un componente tiene un objeto de definición que es un contenedor de los objetos de versión. A su vez, una versión es un contenedor para los conectores, las funciones u otros tipos de componentes.
- Al realizar una implementación en el núcleo de Greengrass, implementará un grupo específico. Una versión de grupo puede contener una versión de cada tipo de componente. Un núcleo es obligatorio, pero los demás se incluyen según sea necesario.
- Las versiones son inmutables, por lo que debe crear nuevas versiones cuando desee realizar cambios.

> **Tip**
>
> Si recibe un error al ejecutar un comando de la AWS CLI, añada el parámetro `--debug` y, a continuación, vuelva a ejecutar el comando para obtener más información sobre el error.

La API de AWS IoT Greengrass le permite crear varias definiciones para un tipo de componente. Por ejemplo, puede crear un objeto `FunctionDefinition` cada vez que cree un parámetro

`FunctionDefinitionVersion`, o bien puede añadir nuevas versiones a una definición existente. Esta flexibilidad le permite personalizar su sistema de administración de versiones.

Requisitos previos

Para completar este tutorial, se necesita lo siguiente:

- Un grupo de Greengrass y un núcleo de Greengrass (v1.9.3 o posterior). Para obtener información acerca de cómo crear un núcleo y un grupo de Greengrass, consulte Empezando con AWS IoT Greengrass. El tutorial de introducción también incluye pasos para instalar el software AWS IoT Greengrass Core.
- Python 3.7 instalado en el dispositivo central de AWS IoT Greengrass.
- AWS IoT Greengrass debe configurarse para admitir secretos locales, como se describe en Requisitos de secretos.

  > **Note**
  >
  > Este requisito incluye permitir el acceso a sus secretos de Secret Manager. Si utiliza el rol de servicio predeterminado de Greengrass, Greengrass tiene permiso para obtener los valores de los secretos cuyos nombres empiecen por greengrass-.

- Un SID de una cuenta de Twilio, un token de autenticación y un número de teléfono habilitado para Twilio. Después de crear un proyecto de Twilio, estos valores están disponibles en el panel del proyecto.

  > **Note**
  >
  > Puede utilizar una cuenta de prueba de Twilio. Si utiliza una cuenta de prueba, debe añadir números de teléfono de destinatarios que no sean de Twilio a una lista de números de teléfono verificados. Para obtener más información, consulte Cómo trabajar con su cuenta de prueba gratuita de Twilio.

- La AWS CLI instalada y configurada en el equipo. Para obtener más información, consulte Instalación de la AWS Command Line Interface y Configuración de la AWS CLI en la Guía del usuario de la AWS Command Line Interface.

Los ejemplos de este tutorial están redactados para Linux y otros sistemas basados en Unix. Si utiliza Windows, consulte Especificación de valores de parámetros para la AWS Command Line Interface para obtener más información sobre las diferencias de sintaxis.

Si el comando contiene una cadena JSON, el tutorial ofrece un ejemplo que tiene el JSON en una sola línea. En algunos sistemas, puede que sea más sencillo editar y ejecutar comandos con este formato.

Paso 1: Crear un secreto de Secrets Manager

En este paso, utilice la API de AWS Secrets Manager para crear un secreto de secreto para su token de autenticación de Twilio.

1. En primer lugar, cree el secreto.

   - Reemplace *twilio-auth-token* por su token de autenticación de Twilio.

   ```
   aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-
   string twilio-auth-token
   ```

   > **Note**
   >
   > De forma predeterminada, el rol de servicio Greengrass permite a AWS IoT Greengrass obtener el valor de los secretos con nombres que comienzan por greengrass-. Para obtener más información, consulte requisitos de secretos.

2. Copie el ARN del secreto de la salida. Puede utilizar esto para crear el recurso de secretos y configurar el conector de Notificaciones de Twilio.

Paso 2: Crear una versión y una definición del recurso

En este paso, utilice la API de AWS IoT Greengrass para crear un recurso de secretos para el secreto de Secrets Manager.

1. Cree una definición de recurso que incluya una versión inicial.

   - Reemplace *secret-arn* por el ARN del secreto que copió en el paso anterior.

JSON Expanded

```
aws greengrass create-resource-definition --name MyGreengrassResources --
initial-version '{
    "Resources": [
        {
            "Id": "TwilioAuthToken",
            "Name": "MyTwilioAuthToken",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "secret-arn"
                }
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-resource-definition \
--name MyGreengrassResources \
--initial-version '{"Resources": [{"Id": "TwilioAuthToken",
 "Name": "MyTwilioAuthToken", "ResourceDataContainer":
 {"SecretsManagerSecretResourceData": {"ARN": "secret-arn"}}}]}'
```

2. Copie el parámetro `LatestVersionArn` de la definición de recurso de la salida. Este valor se usa para añadir la versión de la definición de recurso a la versión de grupo que implementó en el núcleo.

Paso 3: Crear una versión y una definición del conector

En este paso va a configurar parámetros para el conector Notificaciones de Twilio.

1. Cree una definición de conector con una versión inicial.

   - Reemplace *account-sid* por el SID de la cuenta de Twilio.

- Reemplace *secret-arn* por el ARN del secreto del Secrets Manager. El conector utiliza esto para obtener el valor del secreto local.
- Reemplace *phone-number* por el número de teléfono habilitado para Twilio. Twilio lo utiliza para iniciar el mensaje de texto. Esto se puede anular en la carga de mensajes de entrada. Use el siguiente formato: +19999999999.

JSON Expanded

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --
initial-version '{
    "Connectors": [
        {
            "Id": "MyTwilioNotificationsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
            "Parameters": {
                "TWILIO_ACCOUNT_SID": "account-sid",
                "TwilioAuthTokenSecretArn": "secret-arn",
                "TwilioAuthTokenSecretArn-ResourceId": "TwilioAuthToken",
                "DefaultFromPhoneNumber": "phone-number"
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-connector-definition \
--name MyGreengrassConnectors \
--initial-version '{"Connectors": [{"Id": "MyTwilioNotificationsConnector",
 "ConnectorArn": "arn:aws:greengrass:region::/connectors/TwilioNotifications/
versions/4", "Parameters": {"TWILIO_ACCOUNT_SID": "account-sid",
 "TwilioAuthTokenSecretArn": "secret-arn", "TwilioAuthTokenSecretArn-
ResourceId": "TwilioAuthToken", "DefaultFromPhoneNumber": "phone-number"}}]}'
```

> **Note**
>
> `TwilioAuthToken` es el ID que usó en el paso anterior para crear el recurso de secretos.

2. Copie el parámetro `LatestVersionArn` de la definición de conector de la salida. Este valor se usa para añadir la versión de la definición de conector a la versión de grupo que se implementa en el núcleo.

Paso 4: Crear un paquete de implementación de la función de Lambda

Para crear una función de Lambda, primero debe crear un paquete de implementación de funciones de Lambda que contenga el código de la función y las dependencias. Las funciones de Lambda de Greengrass requieren el SDK de AWS IoT Greengrass Core para tareas como la comunicación con los mensajes de MQTT en el entorno principal y el acceso a los secretos locales. En este tutorial se crea una característica de Python para que utilices la versión Python del SDK en el paquete de implementación.

1. Desde la página de descargas del Core SDK AWS IoT Greengrass, descargue el SDK AWS IoT Greengrass básico para Python en su ordenador.
2. Descomprima el paquete descargado para obtener el SDK. El SDK es la carpeta `greengrasssdk`.
3. Guarde la siguiente función de código de Python en un archivo local llamado "`temp_monitor.py`".

```
import greengrasssdk
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
```

```
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. Comprima en un archivo ZIP los siguientes elementos en un archivo denominado "`temp_monitor_python.zip`". Al crear el archivo ZIP, incluya únicamente el código y sus dependencias, no la carpeta donde se encuentran.

   - temp_monitor.py. Lógica de la aplicación.
   - greengrasssdk. Biblioteca necesaria para las funciones de Lambda de Python Greengrass que publican mensajes MQTT.

   Este es el paquete de implementación de la función de Lambda.

Paso 5: Crear una función de Lambda

Ahora, cree una función de Lambda que use el paquete de implementación.

1. Cree un rol de IAM para poder pasar el ARN del rol cuando cree la función.

JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
           "Action": "sts:AssumeRole"
        }
    ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
 "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
 "lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

> **Note**
>
> AWS IoT Greengrass no utiliza este rol, ya que los permisos de las funciones de Lambda de Greengrass se especifican en la función de rol de grupo de Greengrass. En este tutorial, va a crear un rol vacío.

2. Copie la `Arn` del resultado.
3. Utilice la API de AWS Lambda para crear la función TempMonitor. El siguiente comando da por hecho que el archivo ZIP está en el directorio actual.

   - Reemplace *`role-arn`* por el `Arn` que ha copiado.

```
aws lambda create-function \
--function-name TempMonitor \
--zip-file fileb://temp_monitor_python.zip \
--role role-arn \
--handler temp_monitor.function_handler \
```

```
--runtime python3.7
```

4. Publique una versión de la función.

```
aws lambda publish-version --function-name TempMonitor --description 'First
 version'
```

5. Cree un alias a la versión publicada.

   Los grupos de Greengrass pueden hacer referencia a una función de Lambda por versión o alias (recomendado). El uso de un alias facilita la gestión de las actualizaciones del código porque no tiene que cambiar la tabla de suscripción o la definición del grupo cuando se actualiza el código de la función. En su lugar, basta con apuntar el alias a la nueva versión de la función.

   > **Note**
   >
   > AWS IoT Greengrass no admite alias de Lambda; para versiones de $LATEST.

```
aws lambda create-alias --function-name TempMonitor --name GG_TempMonitor --
function-version 1
```

6. Copie la `AliasArn` del resultado. Este valor se usa al configurar la función para AWS IoT Greengrass y al crear una suscripción.

Ahora está listo para configurar la función para AWS IoT Greengrass.

Paso 6: Crear una versión y una definición de la función

Para utilizar una función de Lambda en una AWS IoT Greengrass core, debe crear una versión de la definición de función que haga referencia a la función de Lambda por alias y defina la configuración en el nivel de grupos. Para obtener más información, consulte the section called “Control de la ejecución de la función de Lambda de Greengrass”.

1. Cree una definición de función que incluya una versión inicial.

   - Reemplace *`alias-arn`* por el `AliasArn` que ha copiado al crear el alias.

JSON Expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "TempMonitorFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
                "Executable": "temp_monitor.function_handler",
                "MemorySize": 16000,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "TempMonitorFunction",
 "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
 "temp_monitor.function_handler", "MemorySize": 16000,"Timeout": 5}}]}'
```

2. Copie la `LatestVersionArn` del resultado. Este valor se usa para añadir la versión de la definición de función a la versión de grupo que implementó en el núcleo.
3. Copie la `Id` del resultado. Este valor se usa más tarde al actualizar la función.

Paso 7: Crear una versión y una definición de la suscripción

En este paso añadirá una suscripción que habilita la función de Lambda para enviar datos de entrada al conector. El conector define los temas de MQTT a los que se suscribe, por lo que esta suscripción utiliza uno de los temas. Este es el mismo tema en el que se publica la función de ejemplo.

En este tutorial también creará suscripciones que permitan a la función recibir lecturas de temperatura simuladas de AWS IoT y permitan a AWS IoT recibir información sobre el estado del conector.

1. Cree una definición de suscripción que contenga una versión inicial que incluya las suscripciones.

   - Reemplace *alias-arn* por el `AliasArn` que ha copiado al crear el alias para la función. Utilice este ARN para las dos suscripciones que lo utilizan.

   JSON Expanded

   ```
   aws greengrass create-subscription-definition --initial-version '{
       "Subscriptions": [
           {
               "Id": "TriggerNotification",
               "Source": "alias-arn",
               "Subject": "twilio/txt",
               "Target": "arn:aws:greengrass:region::/connectors/
   TwilioNotifications/versions/4"
           },
           {
               "Id": "TemperatureInput",
               "Source": "cloud",
               "Subject": "temperature/input",
               "Target": "alias-arn"
           },
           {
               "Id": "OutputStatus",
               "Source": "arn:aws:greengrass:region::/connectors/
   TwilioNotifications/versions/4",
               "Subject": "twilio/message/status",
               "Target": "cloud"
           }
       ]
   }'
   ```

   JSON Single-line

   ```
   aws greengrass create-subscription-definition \
   --initial-version '{"Subscriptions": [{"Id": "TriggerNotification", "Source":
    "alias-arn", "Subject": "twilio/txt", "Target": "arn:aws:greengrass:region::/
   connectors/TwilioNotifications/versions/4"},{"Id": "TemperatureInput",
    "Source": "cloud", "Subject": "temperature/input", "Target": "alias-arn"},
   ```

```
{"Id": "OutputStatus", "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4", "Subject": "twilio/message/status", "Target":
 "cloud"}]}'
```

2. Copie la `LatestVersionArn` del resultado. Este valor se usa para añadir la versión de la definición de suscripción a la versión de grupo que implementó en el núcleo.

Paso 8: Crear una versión del grupo

Ahora, puede crear una versión de grupo que contenga todos los elementos que desea implementar. Para ello, cree una versión de grupo que haga referencia a la versión de destino de cada tipo de componente.

En primer lugar, obtenga el ID de grupo y el ARN de la versión de la definición del núcleo. Estos valores son necesarios para crear la versión de grupo.

1. Obtenga el ID del grupo y la última versión del grupo:

   a. Obtenga los ID de la versión de grupo y grupo de Greengrass de destino. En este procedimiento, suponemos que estos son el último grupo y la última versión de grupo. La siguiente consulta devuelve el grupo creado más recientemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups,
    &CreationTimestamp))[0]"
   ```

   También puede hacer la consulta por nombre. No es necesario que los nombres de grupo sean únicos, por lo que podrían devolverse varios grupos.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > También puede encontrar estos valores en la consola de AWS IoT. El ID de grupo se muestra en la página Settings (Configuración) del grupo. Los ID de versión del grupo se muestran en la pestaña Implementaciones del grupo.

   b. Copie el `Id` del grupo de destino de la salida. Puede utilizar esto para obtener la versión de la definición de núcleo y al implementar el grupo.

c. Copie el `LatestVersion` del resultado, que es el ID de la última versión añadida al grupo. Puede utilizar esto para obtener la versión de la definición de núcleo.

2. Obtenga el ARN de la versión de la definición principal:

   a. Obtenga la versión de grupo. En este paso, suponemos que la última versión de grupo incluye una versión de la definición de núcleo.

      - Reemplace *`id-grupo`* con el `Id` que ha copiado para el grupo.
      - Reemplace *`group-version-id`* por el parámetro `LatestVersion` que ha copiado para el grupo.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id group-version-id
```

   b. Copie la `CoreDefinitionVersionArn` del resultado.

3. Cree una versión de grupo.

   - Reemplace *`id-grupo`* con el `Id` que ha copiado para el grupo.
   - Reemplace *`core-definition-version-arn`* por el `CoreDefinitionVersionArn` que ha copiado para la nueva versión de la definición de núcleo.
   - Reemplace *`resource-definition-version-arn`* por el parámetro `LatestVersionArn` que ha copiado para la definición del recurso.
   - Reemplace *`connector-definition-version-arn`* con el valor `LatestVersionArn` que copió para la definición de conector.
   - Reemplace *`function-definition-version-arn`* por el parámetro `LatestVersionArn` que ha copiado para la definición de función.
   - Reemplace *`subscription-definition-version-arn`* por el parámetro `LatestVersionArn` que ha copiado para la definición de suscripción.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--connector-definition-version-arn connector-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
```

```
--subscription-definition-version-arn subscription-definition-version-arn
```

4. Copie el valor de `Version` del resultado. Este es el ID de la versión del grupo. Este valor se usa para implementar la versión de grupo.

Paso 9: Crear una implementación

Implemente el grupo en el dispositivo del núcleo.

1. Asegúrese de que el daemon de AWS IoT Greengrass se está ejecutando en el terminal del dispositivo del núcleo.

   a. Para comprobar si el demonio está en ejecución:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, el demonio está en ejecución.

   b. Iniciar el daemon:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. Cree una implementación de .

   - Reemplace *id-grupo* con el `Id` que ha copiado para el grupo.
   - Reemplace *id-versión-grupo* con el `Version` que ha copiado para el nuevo grupo.

   ```
   aws greengrass create-deployment \
   --deployment-type NewDeployment \
   --group-id group-id \
   --group-version-id group-version-id
   ```

3. Copie la `DeploymentId` del resultado.
4. Obtenga el estado de las implementaciones.

   - Reemplace *id-grupo* con el `Id` que ha copiado para el grupo.

- Reemplace *deployment-id* por el parámetro `DeploymentId` que ha copiado para la implementación.

```
aws greengrass get-deployment-status \
--group-id group-id \
--deployment-id deployment-id
```

Si el estado es `Success`, la implementación fue correcta. Para obtener ayuda sobre la resolución de problemas, consulte Solución de problemas.

Probar la solución

1. En la página de inicio de la consola AWS IoT, elija Pruebas.
2. Para Suscribirse al tema, utilice los siguientes valores y, a continuación, seleccione Suscribirse. El conector Notificaciones de Twilio publica información sobre el estado en este tema.

Propiedad	Valor
Subscription topic	twilio/message/status
Visualización de la carga de MQTT	Display payloads as strings

3. Para Publicar en tema, utilice los siguientes valores y, a continuación, seleccione Publicar para invocar la función.

Propiedad	Valor
Tema	temperature/input
Mensaje	Sustituya *recipient-name* con un nombre y *recipient-phone-number* con el número de teléfono del destinatario del mensaje de texto. Ejemplo: `+12345000000` `{`

Propiedad	Valor

	<pre>"to_name": " recipient-name ",
	"to_number": " recipient-phone-number ",
	"temperature": 31
	}</pre>
	Si utiliza una cuenta de prueba, debe añadir números de teléfono de destinatarios que no sean de Twilio a una lista de números de teléfono verificados. Para obtener más información, consulte Verificar su número de teléfono personal.

Si se ejecuta correctamente, el destinatario recibe el mensaje de texto y la consola muestra el estado `success` de los datos de salida.

Ahora, cambie el valor de `temperature` en el mensaje de entrada a **29** y publíquelo. Dado que es menos de 30, la función TempMonitor no activa un mensaje de Twilio.

Véase también

- Integración con servicios y protocolos mediante conectores
- the section called “conectores de Greengrass proporcionados por AWS”
- the section called “Introducción a los conectores (consola)”
- Comandos de AWS Secrets Manager en la Referencia de comandos de AWS CLI
- Comandos de AWS Identity and Access Management (IAM) en la Referencia de comandos de la AWS CLI.
- Comandos de AWS Lambda en la Referencia de comandos de AWS CLI
- Comandos de AWS IoT Greengrass en la Referencia de comandos de AWS CLI

API RESTful de detección de Greengrass

Todos los dispositivos cliente que se comunican con un AWS IoT Greengrass core deben ser miembros de un grupo de Greengrass. Cada grupo debe tener un núcleo Greengrass. La API de detección permite que los dispositivos puedan recuperar la información necesaria para conectarse a un núcleo de Greengrass que esté en el mismo grupo que el dispositivo cliente. Cuando un dispositivo cliente se pone en línea por primera vez, se puede conectar al servicio de AWS IoT Greengrass y usar la API de detección para encontrar:

- El grupo al que pertenece. Un dispositivo cliente puede ser miembro de hasta 10 grupos.
- La dirección IP y el puerto del núcleo de Greengrass del grupo.
- El certificado de entidad de certificación del grupo, que se puede utilizar para autenticar el dispositivo central de Greengrass.

> **Note**
>
> Los dispositivos cliente también pueden usar los SDKs AWS IoT de dispositivo para detectar información de conectividad de un núcleo de Greengrass. Para obtener más información, consulte AWS IoT SDK de dispositivo.

Para utilizar esta API, envíe solicitudes HTTP al punto de enlace de la API Discovery. Por ejemplo:

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Para obtener una lista de las regiones de Amazon Web Services compatibles y los puntos de conexión para la API de detección AWS IoT Greengrass, consulte los puntos de conexión AWS IoT Greengrass y las cuotas en la sección Referencia general de AWS. Se trata de una API solo del plano de datos. Los puntos de enlace de la administración del grupo y las operaciones de AWS IoT Core son diferentes de los puntos de enlace de la API de detección.

Solicitud

La solicitud contiene los encabezados HTTP estándar y se envía al punto de enlace de detección de Greengrass, como se muestra en los ejemplos siguientes.

El número de puerto depende de si el núcleo está configurado para enviar el tráfico HTTPS a través del puerto 8443 o el puerto 443. Para obtener más información, consulte the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.

Puerto 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/
thing/thing-name
```

Puerto 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/
thing/thing-name
```

Los clientes que se conecten por el puerto 443 deben implementar la extensión TLS de Negociación de Protocolo de Capa de Aplicación (ALPN) y pasar `x-amzn-http-ca` como el `ProtocolName` en el `ProtocolNameList`. Para obtener más información, consulte Protocolos en la Guía para desarrolladores de AWS IoT.

> **Note**
>
> Estos ejemplos utilizan el punto de enlace de Amazon Trust Services (ATS), que se utiliza con certificados de CA raíz de ATS (recomendado). Los puntos de enlace deben coincidir con el tipo de certificado de CA raíz. Para obtener más información, consulte the section called “Los puntos de conexión del servicio deben coincidir con el tipo de certificado”.

Respuesta

En caso de éxito, la respuesta incluye encabezados HTTP estándar, así como el código y el cuerpo siguientes:

```
HTTP 200
BODY: response document
```

Para obtener más información, consulte Ejemplo de documentos de respuesta de detección.

Autorización de detección

Para recuperar la información de conectividad se necesita una política que permita al intermediario ejecutar la acción `greengrass:Discover`. La autenticación mutua TLS con un certificado de cliente es la única forma de autenticación aceptada. A continuación, se muestra una política de ejemplo que permite a un intermediario realizar esta acción:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "greengrass:Discover",
        "Resource": ["arn:aws:iot:us-west-2:123456789012:thing/MyThingName"]
     }]
}
```

Ejemplo de documentos de respuesta de detección

En el siguiente documento, se muestra la respuesta de un dispositivo cliente que es miembro de un grupo que tiene un dispositivo central de Greengrass, un punto de conexión y un certificado de entidad de certificación de grupo:

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
```

```
      ]
    }
  ]
}
```

En el siguiente documento se muestra la respuesta de un dispositivo cliente que es miembro de dos grupos que tiene un dispositivo central de Greengrass, varios puntos de conexión y varios certificados de entidad de certificación de grupo:

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-connection-1-description"
            },
            {
              "id": "core-01-connection-id-2",
              "hostAddress": "core-01-address-2",
              "portNumber": core-01-port-2,
              "metadata": "core-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "gg-group-02-id",
      "Cores": [
         {
            "thingArn":"core-02-thing-arn",
```

```
            "Connectivity" : [
            {
              "id": "core-02-connection-id",
              "hostAddress": "core-02-address",
              "portNumber": core-02-port,
              "metadata": "core-02-connection-1-description"
            }
            ],
            "CAs": [
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
            ]
        }
    ]
    }
}
```

> **Note**
>
> Los grupos de Greengrass deben definir exactamente cada dispositivo central de Greengrass. Cualquier respuesta del servicio de AWS IoT Greengrass que incluya una lista de dispositivos centrales de Greengrass contendrá únicamente un núcleo de Greengrass.

Si ha instalado cURL, puede probar la solicitud de detección. Por ejemplo:

```
$ curl --cert 1a23bc4d56.cert.pem --key 1a23bc4d56.private.key https://greengrass-
ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/thing/MyDevice
{"GGGroups":[{"GGGroupId":"1234a5b6-78cd-901e-2fgh-3i45j6k1789","Cores":
[{"thingArn":"arn:aws:iot:us-west-2:1234567
89012:thing/MyFirstGroup_Core","Connectivity":
[{"Id":"AUTOIP_192.168.1.4_1","HostAddress":"192.168.1.5","PortNumber
":8883,"Metadata":""}]}],"CAs":["-----BEGIN CERTIFICATE-----\ncert-contents\n-----END
 CERTIFICATE-----\n"]}]}
```

Seguridad en AWS IoT Greengrass

La seguridad en la nube de AWS es la mayor prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos que se han diseñado para satisfacer los requisitos de seguridad de las organizaciones más exigentes.

La seguridad es una responsabilidad compartida entre AWS y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta servicios de AWS en Nube de AWS. 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 AWSProgramas de conformidad de . Para obtener información sobre los programas de conformidad que se aplican a AWS IoT Greengrass, consulte Servicios de AWS en el ámbito del 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 los reglamentos aplicables.

Cuando utiliza AWS IoT Greengrass, también es responsable de proteger los dispositivos, la conexión de red local y las claves privadas.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza AWS IoT Greengrass. En los siguientes temas, se le mostrará cómo configurar AWS IoT Greengrass para satisfacer sus objetivos de seguridad y conformidad. También puede aprender a utilizar otros servicios de AWS que lo ayuden a monitorear y proteger los recursos de AWS IoT Greengrass.

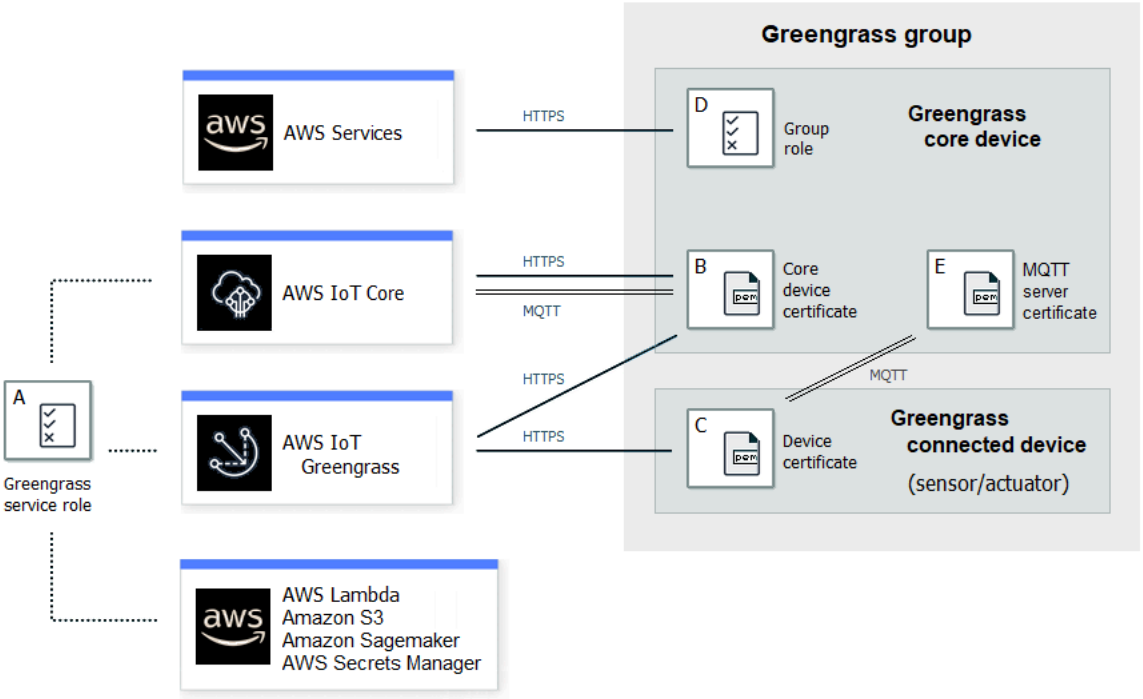
Temas

- Seguridad de la infraestructura en AWS IoT Greengrass
- Configuración y análisis de vulnerabilidades en AWS IoT Greengrass
- AWS IoT Greengrass y puntos de enlace de la VPC de interfaz (AWS PrivateLink)
- Prácticas recomendadas de seguridad para AWS IoT Greengrass

Descripción general de la AWS IoT Greengrass seguridad

AWS IoT Greengrass utiliza certificados, AWS IoT políticas y políticas y funciones de IAM X.509 para proteger las aplicaciones que se ejecutan en los dispositivos de su entorno local de Greengrass.

El siguiente diagrama muestra los componentes del modelo de seguridad: AWS IoT Greengrass



A: rol de servicio de Greengrass

Una función de IAM creada por el cliente que asume AWS IoT Greengrass al acceder a sus AWS recursos desde AWS IoT Core y otros servicios. AWS Lambda AWS Para obtener más información, consulte the section called “Rol de servicio de Greengrass”.

B: certificado del dispositivo del núcleo

Un certificado X.509 utilizado para autenticar un núcleo de Greengrass con y. AWS IoT Core AWS IoT Greengrass Para obtener más información, consulte the section called “Autenticación y autorización de dispositivos”.

C: certificado de dispositivo

Un certificado X.509 utilizado para autenticar un dispositivo cliente, también conocido como dispositivo conectado, con y. AWS IoT Core AWS IoT Greengrass Para obtener más información, consulte the section called “Autenticación y autorización de dispositivos”.

D: rol de grupo

Una función de IAM creada por el cliente que asume AWS IoT Greengrass al llamar a los AWS servicios desde un núcleo de Greengrass.

Esta función se utiliza para especificar los permisos de acceso que necesitan las funciones y conectores de Lambda definidos por el usuario para AWS acceder a los servicios, como DynamoDB. También se usa para permitir exportar las transmisiones del administrador de transmisiones AWS IoT Greengrass a los AWS servicios y escribirlas en los registros. CloudWatch Para obtener más información, consulte the section called “Rol de grupo de Greengrass”.

> **Note**
>
> AWS IoT Greengrass no utiliza la función de ejecución de Lambda que se especifica en la versión AWS Lambda de nube de una función de Lambda.

E - Certificado de servidor MQTT

El certificado utilizado para la autenticación mutua de seguridad de la capa de transporte (TLS) entre un dispositivo del núcleo de Greengrass y dispositivos del cliente en el grupo Greengrass. El certificado está firmado por el certificado de entidad de certificación de grupo, que se almacena en la Nube de AWS.

Flujo de trabajo de conexión de dispositivos

En esta sección se describe cómo los dispositivos cliente se conectan al AWS IoT Greengrass servicio y a los dispositivos principales de Greengrass. Los dispositivos cliente son AWS IoT Core dispositivos registrados que están en el mismo grupo de Greengrass que el dispositivo principal.

- Un dispositivo principal de Greengrass utiliza su certificado de dispositivo, su clave privada y el certificado de CA AWS IoT Core raíz para conectarse al AWS IoT Greengrass servicio. En el dispositivo del núcleo, el objeto `crypto` del archivo de configuración especifica la ruta de archivo de estos elementos.
- El dispositivo del núcleo de Greengrass descarga información de miembros de grupo del servicio de AWS IoT Greengrass .
- Cuando se realiza una implementación en el dispositivo del núcleo de Greengrass, el Administrador de certificados de dispositivo (DCM) gestiona la administración de certificados de servidor local para el dispositivo del núcleo de Greengrass.
- Un dispositivo cliente se conecta al AWS IoT Greengrass servicio mediante su certificado de dispositivo, su clave privada y el certificado de CA AWS IoT Core raíz. Después de realizar la conexión, el dispositivo cliente utiliza el servicio de detección de Greengrass para encontrar la dirección IP de su dispositivo del núcleo de Greengrass. El dispositivo del cliente también descarga el certificado de entidad de certificación de grupo, que se utiliza para la autenticación mutua TLS con el dispositivo del núcleo de Greengrass.
- Un dispositivo cliente intenta conectarse al dispositivo del núcleo de Greengrass, pasándole su certificado de dispositivo y su ID de cliente. Si el ID de cliente coincide con el nombre de objeto del dispositivo cliente y el certificado es válido (parte del grupo de Greengrass), se establece la conexión. De lo contrario, se termina la conexión.

La AWS IoT política para los dispositivos cliente debe conceder el `greengrass:Discover` permiso para permitir que los dispositivos cliente descubran la información de conectividad del núcleo. Para obtener más información sobre esta instrucción de política, consulte the section called “Autorización de detección”.

Configurar la AWS IoT Greengrass seguridad

Para configurar la seguridad de su aplicación Greengrass:

1. Crea cualquier AWS IoT Core cosa para tu dispositivo principal de Greengrass.

2. Genere un par de claves y un certificado de dispositivo para su dispositivo del núcleo de Greengrass.
3. Cree y asocie una política de AWS IoT al certificado del dispositivo. El certificado y la política permiten que el dispositivo principal de Greengrass acceda a los servicios AWS IoT Core y AWS IoT Greengrass los servicios. Para obtener más información, consulte Política mínima de AWS IoT para el dispositivo central.

   > **Note**
   >
   > No se admite el uso de variables de política de objetos (`iot:Connection.Thing.*`) en la AWS IoT política de un dispositivo principal. El núcleo usa el mismo certificado de dispositivo para realizar varias conexiones AWS IoT Core , pero es posible que el ID de cliente de una conexión no coincida exactamente con el nombre del dispositivo principal.

4. Cree un rol de servicio de Greengrass. Esta función de IAM AWS IoT Greengrass le autoriza a acceder a los recursos de otros AWS servicios en su nombre. Esto permite AWS IoT Greengrass realizar tareas esenciales, como recuperar AWS Lambda funciones y gestionar las sombras de los dispositivos.

   Puedes usar el mismo rol de servicio en todos Región de AWS los servidores, pero debe estar asociado a ti Cuenta de AWS en todos los Región de AWS lugares donde lo utilices AWS IoT Greengrass.
5. (Opcional) Cree un rol de grupo de Greengrass. Esta función de IAM concede permiso a las funciones y conectores de Lambda que se ejecutan en un núcleo de Greengrass para llamar a los servicios. AWS Por ejemplo, el conector Kinesis Firehose requiere permiso para escribir registros en una transmisión de entrega de Amazon Data Firehose.

   Solo puede asociar un rol a un grupo de Greengrass.
6. Crea una AWS IoT Core cosa para cada dispositivo que se conecte a tu núcleo de Greengrass.

   > **Note**
   >
   > También puedes usar AWS IoT Core cosas y certificados existentes.

7. Cree certificados de dispositivo, pares de claves y AWS IoT políticas para cada dispositivo que se conecte a su núcleo de Greengrass.

AWS IoT Greengrass principios básicos de seguridad

El núcleo de Greengrass utiliza los siguientes principios de seguridad: AWS IoT cliente, servidor MQTT local y administrador de secretos local. La configuración de estas entidades principales se almacena en el objeto `crypto` en el archivo de configuración `config.json`. Para obtener más información, consulte the section called “Archivo de configuración de AWS IoT Greengrass Core”.

Esta configuración incluye la ruta al archivo de clave privada que la entidad principal utiliza para realizar la autenticación y el cifrado. AWS IoT Greengrass admite dos modos de almacenamiento de claves privadas: basado en hardware o basado en el sistema de archivos (valor predeterminado). Para obtener más información sobre el almacenamiento de claves en los módulos de seguridad de hardware, consulte the section called “Integración de la seguridad de hardware”.

AWS IoT Cliente

El AWS IoT cliente (cliente de IoT) gestiona la comunicación a través de Internet entre el núcleo de Greengrass y. AWS IoT Core AWS IoT Greengrass utiliza certificados X.509 con claves públicas y privadas para la autenticación mutua al establecer conexiones TLS para esta comunicación. Para obtener más información, consulte Certificados X.509 e AWS IoT Core en la Guía para desarrolladores de AWS IoT Core .

El cliente de IoT es compatible con RSA y con los certificados y claves EC. La ruta del certificado y de la clave privada se especifica para la entidad principal `IoTCertificate` en `config.json`.

Servidor MQTT

El servidor MQTT local gestiona la comunicación a través de la red local entre el núcleo de Greengrass y los dispositivos cliente del grupo. AWS IoT Greengrass utiliza certificados X.509 con claves públicas y privadas para la autenticación mutua al establecer conexiones TLS para esta comunicación.

De forma predeterminada, AWS IoT Greengrass genera una clave privada RSA para usted. Para configurar el núcleo de modo que use una clave privada, debe proporcionar la ruta de la clave para la entidad principal `MQTTServerCertificate` en `config.json`. Tiene la responsabilidad de rotar una clave proporcionada por el cliente.

Compatibilidad con clave privada

	Clave RSA	Clave EC
Tipo de clave	Soportado	Soportado

	Clave RSA	Clave EC
Parámetros clave	Longitud mínima de 2048 bits	Curva NIST P-256 o NIST P-384
Formato de disco	PKCS # 1, PKCS # 8	SECG1, PKCS # 8
Versión mínima de GGC	• Usar la clave RSA predeterminada: 1.0 • Especificar una clave RSA: 1.7	• Especificar una clave EC: 1.9

La configuración de la clave privada determina los procesos relacionados. Para ver la lista de conjuntos de cifrado que el núcleo Greengrass admite como servidor, consulte the section called "Compatibilidad con conjuntos de cifrado TLS".

Si no se especifica ninguna clave privada (valor predeterminado)

- AWS IoT Greengrass gira la clave en función de la configuración de rotación.
- El núcleo genera una clave RSA, que se utiliza para generar el certificado.
- El certificado del servidor MQTT tiene una clave pública RSA y una firma SHA-256 RSA.

Si se especifica una clave privada RSA (requiere GGC versión 1.7 o posterior)

- Usted es responsable de rotar la clave.
- El núcleo utiliza la clave especificada para generar el certificado.
- La clave RSA debe tener una longitud mínima de 2048 bits.
- El certificado del servidor MQTT tiene una clave pública RSA y una firma SHA-256 RSA.

Si se especifica una clave privada EC (requiere GGC versión 1.9 o posterior)

- Usted es responsable de rotar la clave.
- El núcleo utiliza la clave especificada para generar el certificado.
- La clave privada EC debe utilizar una curva NIST P-256 o NIST P-384.
- El certificado del servidor MQTT tiene una clave pública EC y una firma SHA-256 RSA.

  El certificado del servidor MQTT presentado por el núcleo tiene una firma SHA-256 RSA, independientemente del tipo de clave. Por este motivo, los clientes deben ser compatibles con la validación de certificados RSA SHA-256 para establecer una conexión segura con el núcleo.

Secrets Manager

El administrador de secretos local gestiona de forma segura las copias locales de los secretos que usted crea en AWS Secrets Manager ellos. Utiliza una clave privada para proteger la clave de datos que se utiliza para cifrar los secretos. Para obtener más información, consulte Implementación de secretos en el núcleo.

De forma predeterminada, se usa la clave privada del cliente de IoT, pero puede especificar otra clave privada para la entidad principal `SecretsManager` en `config.json`. Solo se admite el tipo de claves RSA. Para obtener más información, consulte the section called “Especificación de la clave privada para el cifrado de secretos”.

> **Note**
>
> Actualmente, solo AWS IoT Greengrass admite el mecanismo de relleno PKCS #1 v1.5 para el cifrado y descifrado de secretos locales cuando se utilizan claves privadas basadas en hardware. Si sigue las instrucciones del proveedor para generar manualmente claves privadas basadas en hardware, asegúrese de elegir PKCS #1 v1.5. AWS IoT Greengrass no es compatible con el relleno de cifrado asimétrico óptimo (OAEP).

Compatibilidad con clave privada

	Clave RSA	Clave EC
Tipo de clave	Compatible	No compatible
Parámetros clave	Longitud mínima de 2048 bits	No aplicable
Formato de disco	PKCS # 1, PKCS # 8	No aplicable
Versión de GGC mínima	1.7	No aplicable

Suscripciones administradas en el flujo de trabajo de mensajería de MQTT

AWS IoT Greengrass utiliza una tabla de suscripciones para definir cómo se pueden intercambiar los mensajes MQTT entre los dispositivos, las funciones y los conectores del cliente de un grupo de Greengrass y AWS IoT Core con el servicio paralelo local. Cada suscripción especifica el origen,

el destino y el tema (o asunto) de MQTT sobre el que se envían o reciben los mensajes. AWS IoT Greengrass permite que los mensajes se envíen de un origen a un destino solo si se ha definido la suscripción correspondiente.

Una suscripción define el flujo de mensajes en una sola dirección, del origen al destino. Para admitir el intercambio de mensajes bidireccional, debe crear dos suscripciones, una para cada dirección.

Compatibilidad con conjuntos de cifrado TLS

AWS IoT Greengrass utiliza el modelo de seguridad del AWS IoT Core transporte para cifrar la comunicación con la nube mediante conjuntos de cifrado TLS. Además, AWS IoT Greengrass los datos se cifran cuando están en reposo (en la nube). Para obtener más información sobre la seguridad del AWS IoT Core transporte y los conjuntos de cifrado compatibles, consulte Seguridad del transporte en la Guía para AWS IoT Core desarrolladores.

Conjuntos de cifrado admitidos para las comunicaciones de red locales

Por el contrario AWS IoT Core, el AWS IoT Greengrass núcleo admite los siguientes conjuntos de cifrado TLS de redes locales para los algoritmos de firma de certificados. Todos estos conjuntos de cifrado se admiten cuando las claves privadas se almacenen en el sistema de archivos. Un subgrupo de ellos se admiten cuando el núcleo está configurado para usar módulos de seguridad de hardware (HSM). Para obtener más información, consulte the section called “Entidades de seguridad” y the section called “Integración de la seguridad de hardware”. La tabla también incluye la versión mínima del software AWS IoT Greengrass Core necesaria para la compatibilidad.

	Cifrado	Compatibilidad con HSM	Versión mínima de GGC
TLSv1.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Admitido	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Admitido	1.0
	TLS_ECDHE_RSA_WITH	Admitido	1.0

	Cifrado	Compatibilidad con HSM	Versión mínima de GGC
	_AES_256_ GCM_SHA384		
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	No compatible	1.0
	TLS_RSA_W ITH_AES_1 28_GCM_SHA256	No compatible	1.0
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	No compatible	1.0
	TLS_RSA_W ITH_AES_2 56_GCM_SHA384	No compatible	1.0
	TLS_ECDHE _ECDSA_WI TH_AES_12 8_GCM_SHA256	Compatible	1.9
	TLS_ECDHE _ECDSA_WI TH_AES_25 6_GCM_SHA384	Compatible	1.9
TLSv1.1	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Admitido	1.0
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Admitido	1.0

	Cifrado	Compatibilidad con HSM	Versión mínima de GGC
	TLS_RSA_WITH_AES_128_CBC_SHA	No compatible	1.0
	TLS_RSA_WITH_AES_256_CBC_SHA	No compatible	1.0
TLSv1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Admitido	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Admitido	1.0
	TLS_RSA_WITH_AES_128_CBC_SHA	No compatible	1.0
	TLS_RSA_WITH_AES_256_CBC_SHA	No compatible	1.0

Protección de datos en AWS IoT Greengrass

El modelo de responsabilidad AWS compartida modelo se aplica a la protección de datos en AWS IoT Greengrass. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global que ejecuta todos los Nube de AWS. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Usted también es responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte la sección Privacidad de datos FAQ. Para obtener información sobre la protección de datos en Europa, consulte el modelo de responsabilidad AWS compartida y la entrada del GDPR blog sobre AWS seguridad.

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:

- Utilice la autenticación multifactorial (MFA) con cada cuenta.
- UseSSL/TLSpara comunicarse con AWS los recursos. Necesitamos TLS 1.2 y recomendamos TLS 1.3.
- Configure API y registre la actividad del usuario con AWS CloudTrail.
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita entre FIPS 140 y 3 módulos criptográficos validados para acceder a AWS través de una interfaz de línea de comandos o unaAPI, utilice un FIPS terminal. Para obtener más información sobre los FIPS puntos finales disponibles, consulte la Norma federal de procesamiento de información () 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 trabaja con AWS IoT Greengrass o Servicios de AWS utiliza la consola, API AWS CLI, o. AWS SDKs Cualquier dato que ingrese en etiquetas o campos de formato libre utilizados para nombres se puede emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, le recomendamos encarecidamente que no incluya la información sobre las credenciales URL para validar la solicitud a ese servidor.

Para obtener más información sobre cómo proteger la información confidencial AWS IoT Greengrass, consultethe section called “No registre información confidencial”.

Para obtener más información sobre la protección de datos, consulte el modelo de responsabilidad AWS compartida y la entrada del GDPR blog sobre AWS seguridad.

Temas

- Cifrado de datos
- Integración de la seguridad de hardware

Cifrado de datos

AWS IoT Greengrass utiliza el cifrado para proteger los datos mientras están en tránsito (a través de Internet o red local) y en reposo (almacenados en la Nube de AWS).

Los dispositivos de un entorno de AWS IoT Greengrass suelen recopilar datos que se envían a los servicios de AWS para su posterior procesamiento. Para obtener más información acerca del cifrado de datos en otros servicios de AWS, consulte la documentación de seguridad de ese servicio.

Temas

Cifrado en tránsito

AWS IoT Greengrass tiene tres modos de comunicación donde los datos están en tránsito:

- the section called “Datos en tránsito a través de Internet”. La comunicación entre un núcleo de Greengrass y AWS IoT Greengrass a través de Internet está cifrada.
- the section called “Datos en tránsito a través de la red local”. La comunicación entre un núcleo de Greengrass y los dispositivos cliente a través de una red local está cifrada.
- the section called “Datos del dispositivo central”. La comunicación entre componentes en el dispositivo del núcleo de Greengrass no está cifrada.

Datos en tránsito a través de Internet

AWS IoT Greengrass utiliza Transport Layer Security (TLS) para cifrar toda la comunicación a través de Internet. Todos los datos enviados a la Nube de AWS se envían a través de una conexión TLS utilizando protocolos MQTT o HTTPS, por lo que es seguro de forma predeterminada. AWS IoT Greengrass utiliza el modelo de seguridad de transporte de AWS IoT. Para obtener más información, consulte Seguridad de transporte en la Guía del desarrollador de AWS IoT Core.

Datos en tránsito a través de la red local

AWS IoT Greengrass utiliza TLS para cifrar toda la comunicación a través de la red local entre el núcleo de Greengrass y los dispositivos cliente. Para obtener más información, consulte Conjuntos de cifrado admitidos para las comunicaciones de red locales.

Es su responsabilidad proteger la red local y las claves privadas.

Para los dispositivos de núcleo de Greengrass, es su responsabilidad:

- Mantener el núcleo actualizado con los últimos parches de seguridad.
- Mantenga las bibliotecas del sistema actualizadas con los últimos parches de seguridad.
- Proteja las claves privadas. Para obtener más información, consulte the section called "Administración de claves".

Para los dispositivos cliente, es su responsabilidad:

- Mantener la pila de TLS actualizada.
- Proteja las claves privadas.

Datos del dispositivo central

AWS IoT Greengrass no cifra los datos intercambiados localmente en el dispositivo del núcleo de Greengrass porque los datos no salen del dispositivo. Esto incluye la comunicación entre funciones de Lambda definidas por el usuario, conectores, el SDK de AWS IoT Greengrass Core, y componentes del sistema, como el administrador de secuencias.

Cifrado en reposo

AWS IoT Greengrass almacena los datos:

- the section called "Datos en reposo en la Nube de AWS". Estos datos se cifran.
- the section called "Datos en reposo en el núcleo de Greengrass". Estos datos no están cifrados (excepto las copias locales de sus secretos).

Datos en reposo en la Nube de AWS

AWS IoT Greengrass cifra los datos de los clientes almacenados en la Nube de AWS. Estos datos están protegidos mediante claves de AWS KMS administradas por AWS IoT Greengrass.

Datos en reposo en el núcleo de Greengrass

AWS IoT Greengrass se basa en permisos de archivos Unix y cifrado de disco completo (si está habilitado) para proteger los datos en reposo en el núcleo. Tiene la responsabilidad de proteger el sistema de archivos y el dispositivo.

Sin embargo, AWS IoT Greengrass cifra las copias locales de sus secretos recuperados de AWS Secrets Manager. Para obtener más información, consulte the section called “Cifrado de secretos”.

Administración de claves en el dispositivo del núcleo de Greengrass

Es responsabilidad del cliente garantizar el almacenamiento seguro de claves criptográficas (públicas y privadas) en el dispositivo del núcleo de Greengrass. AWS IoT Greengrass utiliza claves públicas y privadas para las siguientes situaciones:

- La clave de cliente de IoT se utiliza con el certificado IoT para autenticar el protocolo de enlace Transport Layer Security (TLS) cuando un núcleo de Greengrass se conecta a AWS IoT Core. Para obtener más información, consulte the section called “Autenticación y autorización de dispositivos”.

  > **Note**
  >
  > La clave y el certificado también se conocen como clave privada del núcleo y el certificado de dispositivo del núcleo.

- La clave del servidor MQTT se utiliza con el certificado del servidor MQTT para autenticar las conexiones TLS entre los dispositivos del núcleo y cliente. Para obtener más información, consulte the section called “Autenticación y autorización de dispositivos”.
- El administrador de secretos locales también utiliza la clave de cliente de IoT para proteger la clave de datos utilizada para cifrar secretos locales, pero puede proporcionar su propia clave privada. Para obtener más información, consulte the section called “Cifrado de secretos”.

Un núcleo de Greengrass admite el almacenamiento de claves privadas mediante permisos del sistema de archivos, módulos de seguridad de hardware o ambos. Si utiliza claves privadas basadas en el sistema de archivos, es responsable de su almacenamiento seguro en el dispositivo del núcleo.

En un núcleo de Greengrass, la ubicación de sus claves privadas se especifica en la sección `crypto` del archivo `config.json`. Si configura el núcleo para utilizar una clave proporcionada por el cliente para el certificado del servidor MQTT, es su responsabilidad rotar la clave. Para obtener más información, consulte the section called “Entidades de seguridad”.

Para los dispositivos cliente, es su responsabilidad mantener actualizada la pila de TLS y proteger las claves privadas. Las claves privadas se utilizan con certificados de dispositivo para autenticar las conexiones de TLS con el servicio de AWS IoT Greengrass.

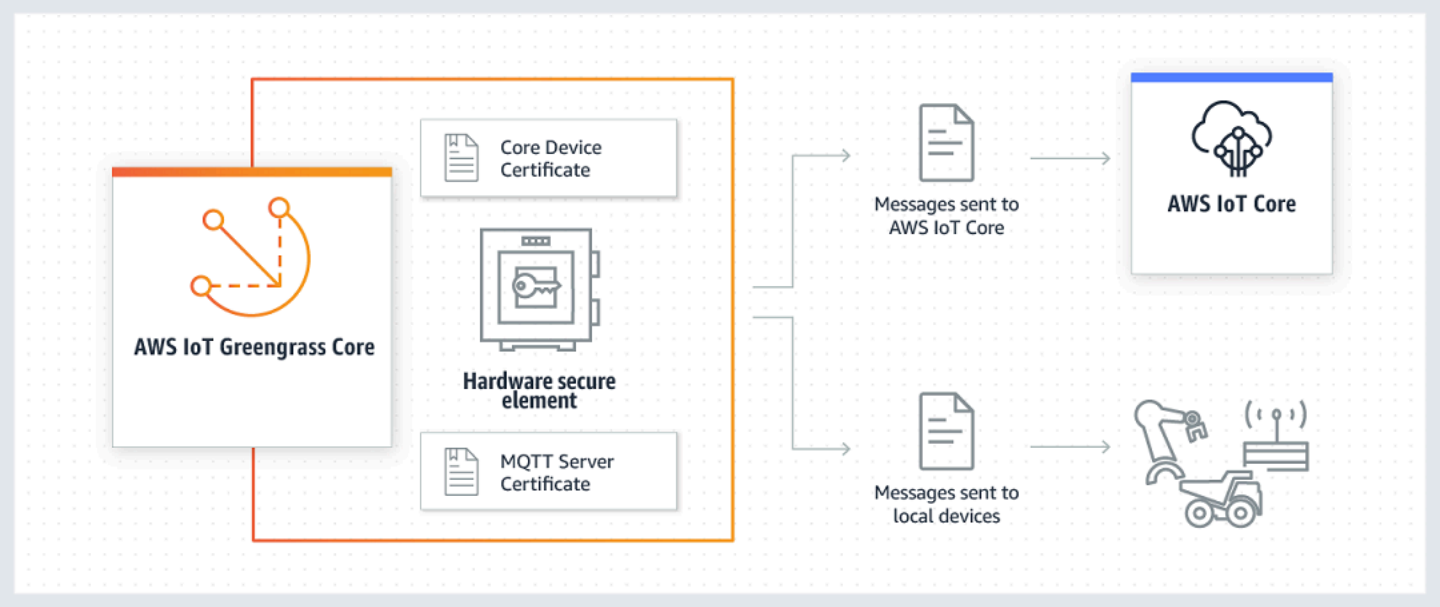
Integración de la seguridad de hardware

Esta función está disponible para AWS IoT Greengrass Core v1.7 y versiones posteriores.

AWS IoT Greengrass admite el uso de módulos de seguridad de hardware (HSM) a través de la interfaz PKCS #11 para el almacenamiento y la descarga seguros de claves privadas. Esto impide que las claves queden expuestas o se dupliquen en el software. Las claves privadas se pueden almacenar de forma segura en módulos de hardwareHSMs, como los módulos de plataforma segura (TPM) u otros elementos criptográficos.

Busque los dispositivos cualificados para esta característica en el AWS Partner Device Catalog.

El siguiente diagrama muestra la arquitectura de seguridad de hardware de un AWS IoT Greengrass núcleo.



En una instalación estándar, AWS IoT Greengrass utiliza dos claves privadas. El componente AWS IoT cliente (cliente de IoT) utiliza una clave durante el protocolo de enlace Transport Layer Security (TLS) cuando se conecta un núcleo de Greengrass a. AWS IoT Core A esta clave también se la conoce como "clave privada del núcleo". La otra clave la usa el MQTT servidor local, lo que permite que los dispositivos Greengrass se comuniquen con el núcleo de Greengrass. Si desea utilizar la seguridad de hardware para ambos componentes, puede utilizar una clave privada compartida o claves privadas independientes. Para obtener más información, consulte the section called “ Prácticas de aprovisionamiento”.

> **Note**
>
> En una instalación estándar, el administrador de secretos local también utiliza la clave de cliente de IoT en su proceso de cifrado, aunque usted puede utilizar su propia clave privada. Debe ser una RSA clave con una longitud mínima de 2048 bits. Para obtener más información, consulte the section called “Especificación de la clave privada para el cifrado de secretos”.

Requisitos

Antes de poder configurar la seguridad de hardware para un núcleo de Greengrass, debe disponer de lo siguiente:

- Un módulo de seguridad de hardware (HSM) que admite la configuración de clave privada de destino para los componentes del cliente de IoT, el MQTT servidor local y el administrador de secretos local. La configuración puede incluir una, dos o tres claves privadas basadas en hardware, en función de si configura los componentes para que compartan claves. Para obtener más información acerca de la compatibilidad con claves privadas, consulte the section called “Entidades de seguridad”.
  - Para RSA las claves: un tamaño de clave de RSA -2048 (o mayor) y un esquema de firma PKCS#1 v1.5.
  - Para llaves EC: una curva NIST P-256 o NIST P-384.

  > **Note**
  >
  > Busque los dispositivos cualificados para esta característica en el AWS Partner Device Catalog.

- Una biblioteca de proveedores PKCS #11 que se puede cargar en tiempo de ejecución (mediante libdl) y que proporciona funciones #11. PKCS
- El módulo de hardware debe poder resolverse mediante una etiqueta de ranura, tal y como se define en la especificación #11. PKCS
- La clave privada debe generarse y cargarse en él mediante las herramientas de HSM aprovisionamiento proporcionadas por el proveedor.
- La clave privada debe poder resolverla la etiqueta de objeto.

- El certificado de dispositivo del núcleo. Se trata de un certificado de cliente de IoT que se corresponde con la clave privada.
- Si utiliza el agente de OTA actualización de Greengrass, debe estar instalada la biblioteca SSLcontenedora Open libp11 PKCS #11. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.

Además, asegúrese de que se cumplen las siguientes condiciones:

- Los certificados de cliente de IoT que están asociados a la clave privada se registran AWS IoT y se activan. Puedes verificarlo en la AWS IoT consola, en Administrar, expandir Todos los dispositivos, elegir Cosas y elegir la pestaña Certificados como elemento principal.
- El software AWS IoT Greengrass principal, versión 1.7 o posterior, está instalado en el dispositivo principal, tal y como se describe en el módulo 2 del tutorial de introducción. Se requiere la versión 1.9 o posterior para usar una clave EC para el MQTT servidor.
- Los certificados se asocian al núcleo de Greengrass. Puedes comprobarlo en la página de administración de la parte principal de la AWS IoT consola.

> **Note**
>
> Actualmente, AWS IoT Greengrass no admite cargar el certificado de CA o el certificado de cliente de IoT directamente desdeHSM. Los certificados se deben cargar como archivos de texto sin formato en el sistema de archivos en una ubicación que pueda leer Greengrass.

Configuración de seguridad de hardware para un AWS IoT Greengrass núcleo

La seguridad de hardware está configurada en el archivo de configuración de Greengrass. Este es el archivo `config.json` ubicado en el directorio `/greengrass-root/config`.

> **Note**
>
> Para ver el proceso de configuración de una HSM configuración mediante una implementación de software puro, consultethe section called “Módulo 7: Simulación de la integración de seguridad del hardware”.

> **Important**
>
> La configuración simulada del ejemplo no proporciona ningún beneficio de seguridad. Su objetivo es permitirle conocer la especificación PKCS #11 y realizar las pruebas iniciales de su software si planea utilizar un software basado HSM en hardware en el futuro.

Para configurar la seguridad del hardware en AWS IoT Greengrass, edite el `crypto` objeto en. `config.json`

Cuando se utiliza la seguridad de hardware, el `crypto` objeto se utiliza para especificar las rutas a los certificados, las claves privadas y los activos de la biblioteca de proveedores PKCS #11 del núcleo, como se muestra en el siguiente ejemplo.

```
"crypto": {
  "PKCS11" : {
    "OpenSSLEngine" : "/path-to-p11-openssl-engine",
    "P11Provider" : "/path-to-pkcs11-provider-so",
    "slotLabel" : "crypto-token-name",
    "slotUserPin" : "crypto-token-user-pin"
  },
  "principals" : {
    "IoTCertificate" : {
      "privateKeyPath" : "pkcs11:object=core-private-key-label;type=private",
      "certificatePath" : "file:///path-to-core-device-certificate"
    },
    "MQTTServerCertificate" : {
      "privateKeyPath" : "pkcs11:object=server-private-key-label;type=private"
    },
    "SecretsManager" : {
      "privateKeyPath": "pkcs11:object=core-private-key-label;type=private"
    }
  },
  "caPath" : "file:///path-to-root-ca"
```

El objeto `crypto` contiene las siguientes propiedades:

Campo	Descripción	Notas
`caPath`	La ruta absoluta a la CA AWS IoT raíz.	Debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. La ruta absoluta al `.so` archivo Open SSL Engine para habilitar la compatibilidad con PKCS #11 en OpenSSL.	Debe ser una ruta a un archivo del sistema de archivos. Esta propiedad es obligatoria si utiliza el agente de OTA actualización de Greengrass con seguridad de hardware. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.
`P11Provider`	La ruta absoluta a la biblioteca PKCS libdl-loadable de la implementación #11.	Debe ser una ruta a un archivo del sistema de archivos.
`slotLabel`	La etiqueta de ranura que se utiliza para identificar el módulo de hardware.	Debe cumplir con las especificaciones de la etiqueta #11PKCS.

Campo	Descripción	Notas
`slotUserPin`	El usuario PIN que se utiliza para autenticar el núcleo de Greengrass en el módulo.	Debe tener permisos suficientes para realizar la firma C_Sign con las claves privadas configuradas.
`principals`		
`IoTCertificate`	El certificado y la clave privada que el núcleo utiliza para realizar solicitudes a AWS IoT.	
`IoTCertificate.privateKeyPath`	La ruta a la clave privada del núcleo.	Para el almacenamiento del sistema de archivos, debe ser un archivo con URI el formato:. `file:///absolute/path/to/file` Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto.
`IoTCertificate.certificatePath`	La ruta absoluta al certificado de dispositivo del núcleo.	Debe ser un archivo con URI el formato:`file:///absolute/path/to/file` .
`MQTTServerCertificate`	Opcional. La clave privada que el núcleo utiliza en combinación con el certificado para actuar como MQTT servidor o puerta de enlace.	

Campo	Descripción	Notas
`MQTTServerCertificate.privateKeyPath`	La ruta a la clave privada MQTT del servidor local.	Utilice este valor para especificar su propia clave privada para el MQTT servidor local. Para el almacenamiento del sistema de archivos, debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto. Si se omite esta propiedad , AWS IoT Greengrass gira la clave en función de la configuración de rotación. Si se especifica, el cliente es responsable de la rotación de la clave.
`SecretsManager`	La clave privada que protege la clave de datos utilizada para el cifrado. Para obtener más información, consulte Implementación de secretos en el núcleo.	

Campo	Descripción	Notas
`SecretsManager .privateKeyPath`	La ruta a la clave privada del administrador de secretos locales.	Solo se admite una RSA clave. Para el almacenamiento del sistema de archivos, debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto. La clave privada debe generarse mediante el mecanismo de relleno PKCS#1 v1.5.

Campo	Descripción	Notas
`caPath`	La ruta absoluta a la CA AWS IoT raíz.	Debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`PKCS11`		

Campo	Descripción	Notas
`OpenSSLEngine`	Opcional. La ruta absoluta al `.so` archivo Open SSL Engine para habilitar la compatibilidad con PKCS #11 en OpenSSL.	Debe ser una ruta a un archivo del sistema de archivos. Esta propiedad es obligatoria si utiliza el agente de OTA actualización de Greengrass con seguridad de hardware. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.
`P11Provider`	La ruta absoluta a la biblioteca PKCS libdl-loadable de la implementación #11.	Debe ser una ruta a un archivo del sistema de archivos.
`slotLabel`	La etiqueta de ranura que se utiliza para identificar el módulo de hardware.	Debe cumplir con las especificaciones de la etiqueta #11PKCS.
`slotUserPin`	El usuario PIN que se utiliza para autenticar el núcleo de Greengrass en el módulo.	Debe tener permisos suficientes para realizar la firma C_Sign con las claves privadas configuradas.
`principals`		
`IoTCertificate`	El certificado y la clave privada que el núcleo utiliza para realizar solicitudes a AWS IoT.	

Campo	Descripción	Notas
`IoTCertificate.privateKeyPath`	La ruta a la clave privada del núcleo.	Para el almacenamiento del sistema de archivos, debe ser un archivo con URI el formato:. `file:///absolute/path/to/file` Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto.
`IoTCertificate.certificatePath`	La ruta absoluta al certificado de dispositivo del núcleo.	Debe ser un archivo con URI el formato:`file:///absolute/path/to/file` .
`MQTTServerCertificate`	Opcional. La clave privada que el núcleo utiliza en combinación con el certificado para actuar como MQTT servidor o puerta de enlace.	

Campo	Descripción	Notas
`MQTTServerCertificate.privateKeyPath`	La ruta a la clave privada MQTT del servidor local.	Utilice este valor para especificar su propia clave privada para el MQTT servidor local. Para el almacenamiento del sistema de archivos, debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto. Si se omite esta propiedad , AWS IoT Greengrass gira la clave en función de la configuración de rotación. Si se especifica, el cliente es responsable de la rotación de la clave.
`SecretsManager`	La clave privada que protege la clave de datos utilizada para el cifrado. Para obtener más información, consulte Implementación de secretos en el núcleo.	

Campo	Descripción	Notas
`SecretsManager .privateKeyPath`	La ruta a la clave privada del administrador de secretos locales.	Solo se admite una RSA clave. Para el almacenamiento del sistema de archivos, debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto. La clave privada debe generarse mediante el mecanismo de relleno PKCS#1 v1.5.

Campo	Descripción	Notas
`caPath`	La ruta absoluta a la CA AWS IoT raíz.	Debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . **Note** Asegúrese de que los puntos de conexión se corresponden con su tipo de certificado.
`PKCS11`		

Campo	Descripción	Notas
`OpenSSLEngine`	Opcional. La ruta absoluta al `.so` archivo Open SSL Engine para habilitar la compatibilidad con PKCS #11 en OpenSSL.	Debe ser una ruta a un archivo del sistema de archivos. Esta propiedad es obligatoria si utiliza el agente de OTA actualización de Greengrass con seguridad de hardware. Para obtener más información, consulte the section called “Configure OTA las actualizaciones”.
`P11Provider`	La ruta absoluta a la biblioteca PKCS libdl-loadable de la implementación #11.	Debe ser una ruta a un archivo del sistema de archivos.
`slotLabel`	La etiqueta de ranura que se utiliza para identificar el módulo de hardware.	Debe cumplir con las especificaciones de la etiqueta #11PKCS.
`slotUserPin`	El usuario PIN que se utiliza para autenticar el núcleo de Greengrass en el módulo.	Debe tener permisos suficientes para realizar la firma C_Sign con las claves privadas configuradas.
`principals`		
`IoTCertificate`	El certificado y la clave privada que el núcleo utiliza para realizar solicitudes a AWS IoT.	

Campo	Descripción	Notas
`IoTCertificate .privateKeyPath`	La ruta a la clave privada del núcleo.	Para el almacenamiento del sistema de archivos, debe ser un archivo con URI el formato:. `file:///absolute/path/to/file` Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto.
`IoTCertificate .certificatePath`	La ruta absoluta al certificado de dispositivo del núcleo.	Debe ser un archivo con URI el formato:`file:///absolute/path/to/file` .
`MQTTServerCertificate`	Opcional. La clave privada que el núcleo utiliza en combinación con el certificado para actuar como MQTT servidor o puerta de enlace.	

Campo	Descripción	Notas
`MQTTServerCertificate .privateKeyPath`	La ruta a la clave privada MQTT del servidor local.	Utilice este valor para especificar su propia clave privada para el MQTT servidor local. Para el almacenamiento del sistema de archivos, debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto. Si se omite esta propiedad , AWS IoT Greengrass gira la clave en función de la configuración de rotación. Si se especifica, el cliente es responsable de la rotación de la clave.
`SecretsManager`	La clave privada que protege la clave de datos utilizada para el cifrado. Para obtener más información, consulte Implementación de secretos en el núcleo.	

Campo	Descripción	Notas
`SecretsManager .privateKeyPath`	La ruta a la clave privada del administrador de secretos locales.	Solo se admite una RSA clave. Para el almacenamiento del sistema de archivos, debe ser un archivo URI con el formato:`file:///`*`absolute/path/to/file`* . Para el HSM almacenamiento, debe ser una ruta RFC7512 PKCS #11 que especifique la etiqueta del objeto. La clave privada debe generarse mediante el mecanismo de relleno PKCS#1 v1.5.

Prácticas de aprovisionamiento para la seguridad del hardware AWS IoT Greengrass

A continuación, se muestran las prácticas de aprovisionamiento relacionadas con el rendimiento y la seguridad.

Seguridad

- Genere claves privadas directamente en él HSM mediante el generador de números aleatorios de hardware interno.

  > **Note**
  >
  > Si configura las claves privadas para utilizarlas con esta función (siguiendo las instrucciones proporcionadas por el fabricante del hardware), tenga en cuenta que AWS IoT Greengrass actualmente solo admite el mecanismo de relleno de la PKCS1 versión 1.5 para el cifrado y descifrado de los secretos locales. AWS IoT Greengrass no admite el relleno de cifrado asimétrico óptimo (). OAEP

- Configure las claves privadas para prohibir la exportación.

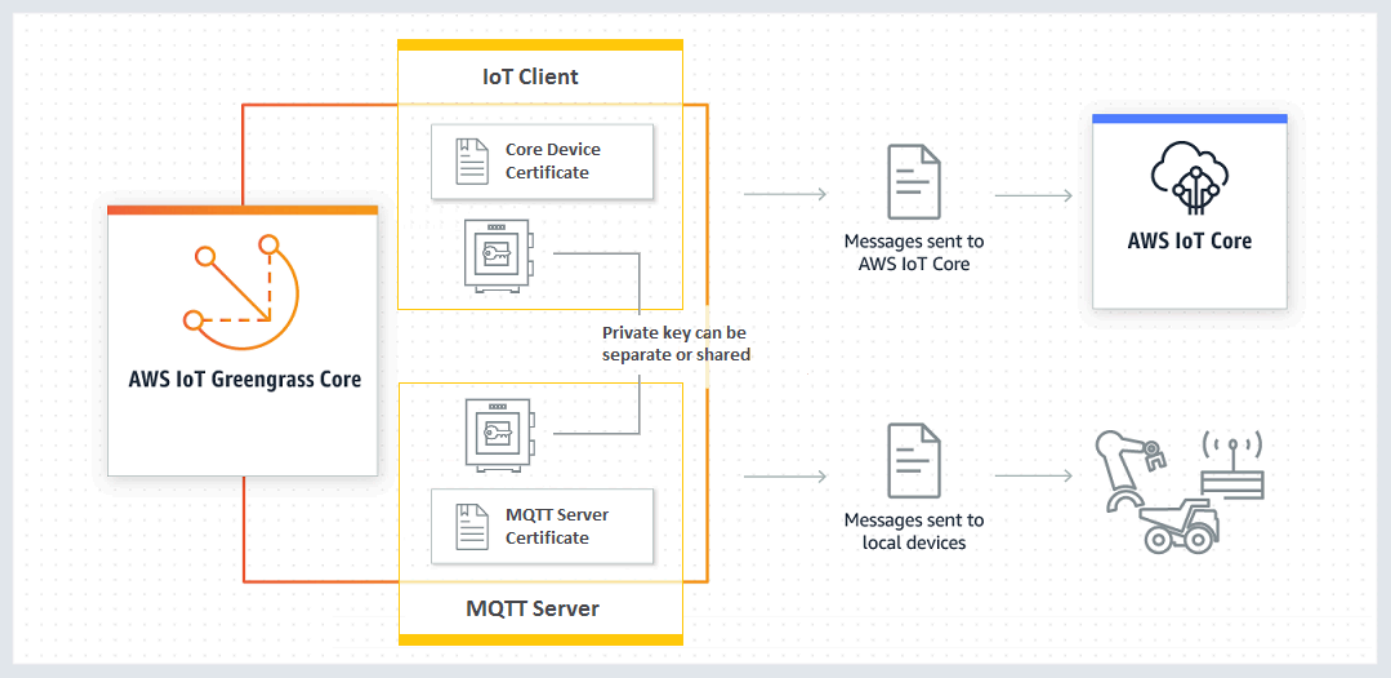
- Utilice la herramienta de aprovisionamiento proporcionada por el proveedor del hardware para generar una solicitud de firma de certificado (CSR) con la clave privada protegida por hardware y, a continuación, utilice la AWS IoT consola para generar un certificado de cliente.

> **Note**
>
> La práctica de rotar las claves no se aplica cuando las claves privadas se generan en un. HSM

Rendimiento

El siguiente diagrama muestra el componente de cliente de IoT y el MQTT servidor local en el AWS IoT Greengrass núcleo. Si desea usar una HSM configuración para ambos componentes, puede usar la misma clave privada o claves privadas independientes. Si utiliza claves independientes, deben almacenarse en la misma ranura.

> **Note**
>
> AWS IoT Greengrass no impone ningún límite a la cantidad de claves que se almacenan en elHSM, por lo que puede almacenar claves privadas para los componentes del cliente, el MQTT servidor y el administrador de secretos de IoT. Sin embargo, algunos HSM proveedores pueden imponer límites a la cantidad de claves que puedes almacenar en una ranura.



En general, la clave de cliente de IoT no se usa con mucha frecuencia porque el software AWS IoT Greengrass Core mantiene conexiones de larga duración a la nube. Sin embargo, la clave MQTT del servidor se usa cada vez que un dispositivo Greengrass se conecta al núcleo. Estas interacciones directamente afectan al rendimiento.

Cuando la clave MQTT del servidor está almacenada en elHSM, la velocidad a la que se pueden conectar los dispositivos depende del número de operaciones de RSA firma por segundo que HSM puedan realizar. Por ejemplo, si HSM se tarda 300 milisegundos en realizar una firma RSASSA PKCS1 -v1.5 en una clave privada RSA -2048, solo se pueden conectar tres dispositivos al núcleo de Greengrass por segundo. Una vez realizadas las conexiones, deja de usarse y HSM se aplican las cuotas estándar para ello. AWS IoT Greengrass

Para mitigar los cuellos de botella en el rendimiento, puede almacenar la clave privada del MQTT servidor en el sistema de archivos en lugar de en el. HSM Con esta configuración, el MQTT servidor se comporta como si la seguridad del hardware no estuviera habilitada.

AWS IoT Greengrass admite múltiples configuraciones de almacenamiento de claves para los componentes de cliente y MQTT servidor de IoT, por lo que puede optimizarlas según sus requisitos de seguridad y rendimiento. En la siguiente tabla se incluyen ejemplos de configuraciones.

Configuración	Clave de IoT	MQTTclave	Rendimiento
HSMClave compartida	HSM: Clave A	HSM: Clave A	Limitado por: HSM o CPU
HSMLlaves separadas	HSM: Clave A	HSM: Tecla B	Limitado por: HSM o CPU
HSMsolo para IoT	HSM: Clave A	Sistema de archivos: Clave B	Limitado por la CPU
Legacy	Sistema de archivos: Clave A	Sistema de archivos: Clave B	Limitado por el CPU

Para configurar el núcleo de Greengrass para que utilice claves basadas en el sistema de archivos para el MQTT servidor, omita la `principals.MQTTServerCertificate` sección de `config.json` (o especifique una ruta de acceso a la clave basada en archivos si no utiliza la clave predeterminada generada por). AWS IoT Greengrass El objeto `crypto` resultante tiene este aspecto:

```
"crypto": {
  "PKCS11": {
    "OpenSSLEngine": "...",
    "P11Provider": "...",
    "slotLabel": "...",
    "slotUserPin": "..."
  },
  "principals": {
    "IoTCertificate": {
      "privateKeyPath": "...",
      "certificatePath": "..."
    },
    "SecretsManager": {
      "privateKeyPath": "..."
    }
  },
  "caPath" : "..."
}
```

Compatibilidad de los conjuntos de cifrado con la integración de seguridad del hardware

AWS IoT Greengrass admite un conjunto de conjuntos de cifrado cuando el núcleo está configurado para garantizar la seguridad del hardware. Se trata de un subconjunto de los conjuntos de cifrado que se admiten cuando el núcleo se ha configurado para utilizar la seguridad basada en archivos. Para obtener más información, consulte the section called “Compatibilidad con conjuntos de cifrado TLS”.

> **Note**
>
> Cuando se conecte al núcleo de Greengrass desde dispositivos Greengrass a través de la red local, asegúrese de utilizar uno de los conjuntos de cifrado compatibles para realizar la conexión. TLS

Configure la compatibilidad con las actualizaciones over-the-air

Para habilitar las actualizaciones over-the-air (OTA) del software AWS IoT Greengrass principal al utilizar la seguridad del hardware, debe instalar la biblioteca contenedora OpenSC PKCSlibp11 #11 y editar el archivo de configuración de Greengrass. Para obtener más información sobre las actualizaciones, consulte. OTA Actualizaciones de OTA para el software AWS IoT Greengrass Core

1. Detenga el daemon de Greengrass.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd stop
   ```

   > **Note**
   >
   > *greengrass-root* representa la ruta en la que se instala el software AWS IoT Greengrass principal en el dispositivo. Normalmente, este es el directorio `/greengrass`.

2. Instale el SSL motor Open. Se admiten Open SSL 1.0 o 1.1.

   ```
   sudo apt-get install libengine-pkcs11-openssl
   ```

3. Busque la ruta al SSL motor Open (`libpkcs11.so`) en su sistema:

a. Obtenga la lista de los paquetes instalados para la biblioteca.

```
sudo dpkg -L libengine-pkcs11-openssl
```

El archivo `libpkcs11.so` se encuentra en el directorio `engines`.

b. Copie la ruta completa al archivo (por ejemplo, `/usr/lib/ssl/engines/libpkcs11.so`).

4. Abra el archivo de configuración de Greengrass. Este es el archivo `config.json` del directorio */greengrass-root*`/config`.

5. Para la propiedad `OpenSSLEngine`, escriba la ruta al archivo `libpkcs11.so`.

```
{
 "crypto": {
   "caPath" : "file:///path-to-root-ca",
   "PKCS11" : {
     "OpenSSLEngine" : "/path-to-p11-openssl-engine",
     "P11Provider" : "/path-to-pkcs11-provider-so",
     "slotLabel" : "crypto-token-name",
     "slotUserPin" : "crypto-token-user-pin"
    },
    ...
  }
  ...
}
```

> **Note**
>
> Si la propiedad `OpenSSLEngine` no existe en el objeto `PKCS11`, añádalo.

6. Iniciar el daemon de Greengrass.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

Compatibilidad con versiones anteriores del software AWS IoT Greengrass principal

El software AWS IoT Greengrass Core, con soporte de seguridad de hardware, es totalmente compatible con versiones anteriores de `config.json` los archivos generados para la versión 1.6 y versiones anteriores. Si el `crypto` objeto no está presente en el archivo `config.json` de configuración, AWS IoT Greengrass utiliza las propiedades y `coreThing.caPath` basadas en el archivo`coreThing.certPath`. `coreThing.keyPath` Esta compatibilidad con versiones anteriores se aplica a OTA las actualizaciones de Greengrass, que no sobrescriben la configuración basada en archivos especificada en. `config.json`

Hardware sin soporte para #11 PKCS

La biblioteca PKCS #11 normalmente la proporciona el proveedor de hardware o es de código abierto. Por ejemplo, con un hardware que cumpla con los estándares (por ejemplo, TPM1 .2), podría ser posible utilizar el software de código abierto existente. Sin embargo, si su hardware no tiene una implementación de biblioteca PKCS #11 correspondiente, o si desea crear un proveedor PKCS #11 personalizado, póngase en contacto con su representante de AWS Enterprise Support si tiene preguntas relacionadas con la integración.

Véase también

- PKCSGuía de uso de la interfaz de token criptográfico #11, versión 2.40. Editado por John Leiseboer y Robert Griffin. 16 de noviembre de 2014. OASISNota del Comité 02. http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html. Última versión: http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.
- RFC7512
- PKCS#1: RSA Cifrado, versión 1.5

Autenticación y autorización de dispositivos en AWS IoT Greengrass

Los dispositivos en entornos de AWS IoT Greengrass utilizan certificados X.509 para la autenticación y políticas de AWS IoT para la autorización. Los certificados y las políticas permiten que los dispositivos se conecten de forma segura entre sí, con AWS IoT Core y AWS IoT Greengrass.

Los certificados X.509 son certificados digitales que utilizan el estándar de infraestructura de clave pública X.509 para asociar una clave pública a una identidad contenida en un certificado. Una

entidad de confianza conocida como entidad de certificación (CA) emite los certificados X.509. La CA administra uno o varios certificados especiales llamados certificados de CA, que utiliza para generar certificados X.509. Solo la entidad de certificación tiene acceso a los certificados de entidad de certificación.

Las políticas de AWS IoT definen el conjunto de operaciones permitidas para los dispositivos de AWS IoT. Específicamente, permiten y deniegan el acceso a operaciones de plano de datos de AWS IoT Core y AWS IoT Greengrass, como la publicación de mensajes MQTT y la recuperación de sombras de dispositivos.

Todos los dispositivos requieren una entrada en el registro de AWS IoT Core y un certificado X.509 activado con una política de AWS IoT asociada. Los dispositivos se dividen en dos categorías:

- Núcleos de Greengrass. Los dispositivos del núcleo de Greengrass utilizan certificados y políticas de AWS IoT para conectarse de forma segura a AWS IoT Core. Los certificados y las políticas también permiten que AWS IoT Greengrass implemente información de configuración, funciones de Lambda, conectores y suscripciones administradas en dispositivos principales.
- Dispositivos cliente. Los dispositivos cliente (también denominados dispositivos conectados, dispositivos Greengrass o dispositivos) son dispositivos que se conectan a un núcleo de Greengrass a través de MQTT. Utilizan certificados y políticas para conectarse a AWS IoT Core y al servicio AWS IoT Greengrass. Esto permite a los dispositivos del cliente utilizar el servicio de detección de AWS IoT Greengrass para buscar un dispositivo del núcleo y conectarse a él. Un dispositivo de cliente utiliza el mismo certificado para conectarse a la gateway del dispositivo de AWS IoT Core y al dispositivo principal. Los dispositivos de cliente también utilizan información de detección para la autenticación mutua con el dispositivo principal. Para obtener más información, consulte the section called “Flujo de trabajo de conexión de dispositivos” y the section called “Administre la autenticación de dispositivos con el núcleo de Greengrass”.

Certificados X.509

La comunicación entre dispositivos principales y de cliente, y entre dispositivos y AWS IoT Core o AWS IoT Greengrass debe autenticarse. Esta autenticación mutua se basa en certificados de dispositivo X.509 registrados y claves criptográficas.

En un entorno de AWS IoT Greengrass, los dispositivos utilizan certificados con claves públicas y privadas para las siguientes conexiones de Transport Layer Security (TLS):

- El componente de cliente de AWS IoT en el núcleo de Greengrass que se conecta a AWS IoT Core y a AWS IoT Greengrass través de Internet.
- Los dispositivos cliente se conectan a AWS IoT Greengrass para obtener información básica de detección a través de Internet.
- El componente de servidor MQTT en el núcleo de Greengrass que se conecta a los dispositivos cliente en el grupo a través de la red local.

El dispositivo principal de AWS IoT Greengrass almacena los certificados en dos ubicaciones:

- Certificado del dispositivo del núcleo en */greengrass-root*/certs. Normalmente, el certificado de dispositivo del núcleo se denomina *hash*.cert.pem (por ejemplo, 86c84488a5.cert.pem). Este certificado es utilizado por el cliente de AWS IoT para la autenticación mutua cuando el núcleo se conecta a los servicios AWS IoT Core y AWS IoT Greengrass.
- Certificado de servidor MQTT en */greengrass-root*/ggc/var/state/server. El certificado del servidor MQTT se denomina server.crt. Este certificado se utiliza para la autenticación mutua entre el servidor MQTT local (en el núcleo de Greengrass) y los dispositivos Greengrass.

> **Note**
>
> *greengrass-root* representa la ruta donde está instalado el software de AWS IoT Greengrass Core en su dispositivo. Normalmente, este es el directorio /greengrass.

Para obtener más información, consulte the section called “Entidades de seguridad”.

Certificados de entidad de certificación (CA)

Los dispositivos de núcleo y los dispositivos cliente descargan un certificado de entidad de certificación raíz utilizado para la autenticación con los servicios de AWS IoT Core y AWS IoT Greengrass. Le recomendamos que utilice un certificado de entidad de certificación raíz de Amazon Trust Services (ATS), como Amazon Root CA 1. Para obtener más información, consulte Certificados de CA para autenticación de servidor en la Guía del desarrollador de AWS IoT Core.

> **Note**
>
> El tipo de certificado de CA raíz debe coincidir con su punto de enlace. Utilice un certificado de CA raíz de ATS con un punto final de ATS (preferido) o un certificado de CA VeriSign

raíz con un punto de conexión heredado. Solo algunas regiones de Amazon Web Services admiten puntos de conexión heredados. Para obtener más información, consulte the section called “Los puntos de conexión del servicio deben coincidir con el tipo de certificado”.

Los dispositivos cliente también descargan el certificado de entidad de certificación del grupo de Greengrass. Esto se utiliza para validar el certificado del servidor MQTT en el núcleo de Greengrass durante la autenticación mutua. Para obtener más información, consulte the section called “Flujo de trabajo de conexión de dispositivos”. El vencimiento predeterminado del certificado del servidor MQTT es de siete días.

Rotación de certificados en el servidor MQTT local

Los dispositivos cliente utilizan el certificado de servidor MQTT local para la autenticación mutua con el dispositivo del núcleo de Greengrass. De forma predeterminada, este certificado caduca en siete días. Este período limitado se basa en las prácticas recomendadas de seguridad. El certificado del servidor MQTT está firmado por el certificado de entidad de certificación del grupo, que se almacena en la nube.

Para que se produzca rotación de certificados, su dispositivo principal de Greengrass debe estar en línea y ser capaz de acceder directamente al servicio de AWS IoT Greengrass de forma periódica. Cuando vence el certificado, el dispositivo principal de Greengrass intenta conectarse al servicio de AWS IoT Greengrass para obtener un nuevo certificado. Si la conexión se realiza correctamente, el dispositivo central descarga un nuevo certificado de servidor de MQTT y reinicia el servicio de MQTT local. En este momento, se desconectan todos los dispositivos cliente que están conectados al núcleo. Si el núcleo del dispositivo está sin conexión en el momento en que vence el certificado, no recibe el certificado de sustitución. Cualquier nuevo intento de conectarse al dispositivo central se rechazará. Las conexiones existentes no se ven afectadas. Los dispositivos cliente no pueden conectarse al dispositivo principal hasta que la conexión al servicio de AWS IoT Greengrass se restaura y se puede descargar un nuevo certificado del servidor de MQTT.

Puede configurar el vencimiento en cualquier valor comprendido entre 7 y 30 días, en función de sus necesidades. Una rotación más frecuente requiere conexiones a la nube más frecuentes. Una rotación menos frecuente puede suponer un riesgo para la seguridad. Si desea definir el vencimiento del certificado en un valor superior a 30 días, póngase en contacto con AWS Support.

En la consola de AWS IoT, puede administrar el certificado en la página Configuración del grupo. En la AWS IoT Greengrass API, puedes usar la UpdateGroupCertificateConfigurationacción.

Cuando el certificado de servidor de MQTT vence, se produce un error en cualquier intento de validar el certificado. Los dispositivos de cliente debe poder detectar el error y terminar la conexión.

Políticas de AWS IoT para operaciones de plano de datos

Utilice las políticas de AWS IoT para autorizar el acceso al plano de datos de AWS IoT Core y AWS IoT Greengrass. El plano de datos de AWS IoT Core consta de operaciones para dispositivos, usuarios y aplicaciones, como conectarse a AWS IoT Core y suscribirse a temas. El plano de datos de AWS IoT Greengrass consta de operaciones para dispositivos de Greengrass, como la recuperación de implementaciones y la actualización de la información de conectividad.

Una política de AWS IoT es un documento JSON que es similar a una política de IAM. Contiene una o varias instrucciones de política que especifican las siguientes propiedades:

- `Effect`. El modo de acceso, que puede ser `Allow` o `Deny`.
- `Action`. La lista de acciones permitidas o denegadas por la política.
- `Resource`. La lista de recursos en los que se permite o deniega la acción.

Las políticas de AWS IoT admiten * como caracteres comodín y tratan los caracteres comodín MQTT (+ y #) como cadenas literales. Para obtener más información sobre el comodín *, consulte Uso del comodín en los ARN de los recursos en la Guía del usuario de AWS Identity and Access Management.

Para obtener más información, consulte Políticas de AWS IoT y Acciones de política de AWS IoT en la Guía del desarrollador de AWS IoT Core.

> **Note**
>
> AWS IoT Core permite adjuntar políticas de AWS IoT a grupos de objetos a fin de definir los permisos para grupos de dispositivos. Las políticas de grupos de objetos no permiten el acceso a las operaciones del plano de datos de AWS IoT Greengrass. Para permitir que una objeto acceda a una operación del plano de datos de AWS IoT Greengrass, añada el permiso a una política de AWS IoT que adjunta al certificado del objeto.

Acciones de política de AWS IoT Greengrass

Acciones del núcleo de Greengrass

AWS IoT Greengrass define las siguientes acciones de política que los dispositivos del núcleo de Greengrass pueden usar en las políticas de AWS IoT:

`greengrass:AssumeRoleForGroup`

Permiso para que un dispositivo del núcleo de Greengrass recupere credenciales mediante la función de Lambda del sistema de servicio de intercambio de token (TES). Los permisos vinculados a las credenciales recuperadas se basan en la política asociada al rol de grupo configurado.

Este permiso se comprueba cuando un dispositivo del núcleo de Greengrass intenta recuperar credenciales (suponiendo que las credenciales no se almacenan en caché localmente).

`greengrass:CreateCertificate`

Permiso para que un dispositivo del núcleo de Greengrass cree su propio certificado de servidor.

Este permiso se comprueba cuando un dispositivo del núcleo de Greengrass crea un certificado. Los dispositivos del núcleo de Greengrass intentan crear un certificado de servidor al ejecutarse por primera vez, cuando cambia la información de conectividad del núcleo y en los períodos de rotación designados.

`greengrass:GetConnectivityInfo`

Permiso para que un dispositivo del núcleo de Greengrass recupere su propia información de conectividad.

Este permiso se comprueba cuando un dispositivo del núcleo de Greengrass intenta recuperar su información de conectividad desde AWS IoT Core.

`greengrass:GetDeployment`

Permiso para que un dispositivo del núcleo de Greengrass recupere implementaciones.

Este permiso se comprueba cuando un dispositivo del núcleo de Greengrass intenta recuperar implementaciones y estados de implementación desde la nube.

`greengrass:GetDeploymentArtifacts`

Permiso para que un dispositivo del núcleo de Greengrass recupere artefactos de implementación como información de grupo o funciones de Lambda.

Este permiso se comprueba cuando un dispositivo del núcleo de Greengrass recibe una implementación y, a continuación, intenta recuperar artefactos de implementación.

`greengrass:UpdateConnectivityInfo`

Permiso para que un dispositivo del núcleo de Greengrass actualice su propia información de conectividad con información IP o nombre de host.

Este permiso se comprueba cuando un dispositivo del núcleo de Greengrass intenta actualizar su información de conectividad en la nube.

`greengrass:UpdateCoreDeploymentStatus`

Permiso para que un dispositivo del núcleo de Greengrass actualice el estado de una implementación.

Este permiso se comprueba cuando un dispositivo del núcleo de Greengrass recibe una implementación y, a continuación, intenta actualizar el estado de implementación.

Acciones de dispositivos de Greengrass

AWS IoT Greengrass define la siguiente acción de política que los dispositivos de cliente pueden usar en las políticas de AWS IoT:

`greengrass:Discover`

Permiso para que un dispositivo de cliente utilice la API de detección para recuperar la información de conectividad cenral de su grupo y la autoridad de certificación de grupo.

Este permiso se comprueba cuando un dispositivo de cliente llama a la API de detección con autenticación mutua TLS.

Política mínima de AWS IoT para el dispositivo central de AWS IoT Greengrass

La siguiente política de ejemplo incluye un conjunto mínimo de acciones necesario para respaldar la funcionalidad básica de Greengrass para su dispositivo del núcleo.

- La política muestra los temas de MQTT y los filtros de tema en los que el dispositivo del núcleo puede publicar mensajes, suscribirse y recibir mensajes, incluidos temas utilizados para estado

de sombra. Para admitir el intercambio de mensajes entre AWS IoT Core, funciones de Lambda, conectores y dispositivos cliente en el grupo de Greengrass, especifique los temas y filtros de temas que desea permitir. Para obtener más información, consulte Ejemplos de política de publicación/suscripción en la Guía del desarrollador de AWS IoT Core.

- La política incluye una sección que permite a AWS IoT Core obtener, actualizar y eliminarla sombra del dispositivo de núcleo. Para permitir la sincronización de sombras para dispositivos cliente conectados en el grupo de Greengrass, especifique los nombres de recursos de Amazon (ARN) de destino en la lista `Resource` (por ejemplo, `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`device-name`*).
- No se permite utilizar variables de políticas de objetos (`iot:Connection.Thing.*`) en la política de AWS IoT de un dispositivo del núcleo. El dispositivo principal utiliza el mismo certificado del dispositivo para realizar varias conexiones a AWS IoT Core, pero puede ocurrir que el ID de cliente de una conexión no coincida exactamente con el nombre del objeto principal.
- Para el permiso `greengrass:UpdateCoreDeploymentStatus`, el segmento final de la ARN `Resource` es el ARN codificado en URL del dispositivo de núcleo.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:client/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-name-*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:AssumeRoleForGroup",
                "greengrass:CreateCertificate"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeployment"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeploymentArtifacts"
            ],
```

```
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:UpdateCoreDeploymentStatus"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*/cores/arn%3Aaws%3Aiot%3Aregion%3Aaccount-id%3Athing%2Fcore-name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        }
    ]
}
```

Note

Las políticas de AWS IoT para dispositivos cliente normalmente requieren permisos similares para acciones `iot:Connect`, `iot:Publish`, `iot:Receive` y `iot:Subscribe`. Para permitir que un dispositivo de cliente detecte automáticamente la información de conectividad de los núcleos de los grupos de Greengrass a los que pertenece el dispositivo, la política de AWS IoT de un dispositivo de cliente debe incluir la acción `greengrass:Discover`. En la sección `Resource`, especifique el ARN del dispositivo de cliente, no el ARN del dispositivo principal de Greengrass. Por ejemplo:

```
{
    "Effect": "Allow",
    "Action": [
        "greengrass:Discover"
```

```
    ],
    "Resource": [
        "arn:aws:iot:region:account-id:thing/device-name"
    ]
}
```

La política de AWS IoT para los dispositivos cliente normalmente no requiere permisos para acciones `iot:GetThingShadow`, `iot:UpdateThingShadow` o `iot:DeleteThingShadow`, ya que el núcleo de Greengrass controla las operaciones de sincronización de sombras para los dispositivos de clientes. En este caso, asegúrese de que la sección `Resource` para acciones de sombra en la política central de AWS IoT incluya los ARN de los dispositivos cliente.

En la consola de AWS IoT, puede ver y editar la política que está asociada su certificado principal.

1. En el panel de navegación, en Administrar, expanda Todos los dispositivos y, a continuación, elija Objetos.
2. Elija su núcleo.
3. En la página de configuración del núcleo, seleccione la pestaña Certificados.
4. En la pestaña Certificados, elija su certificado.
5. En la página de configuración del certificado, elija Policies (Políticas) y, a continuación, seleccione la política.

   Si desea editar la política, elija Editar versión activa.
6. Revise la política y agregue, elimine o edite los permisos según sea necesario.
7. Para establecer una nueva versión de la política como la versión activa, en Estado de la versión de la política, seleccione Establecer la versión editada como la versión activa de esta política.
8. Seleccione Guardar como versión nueva.

Administración de identidad y acceso para AWS IoT Greengrass

AWS Identity and Access Management (IAM) es una Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a AWS los recursos. IAMlos administradores controlan quién

puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar AWS IoT Greengrass los recursos. IAMes una Servicio de AWS que puede utilizar sin coste adicional.

> **Note**
>
> En este tema se describen IAM los conceptos y las características. Para obtener información sobre IAM las funciones compatibles con AWS IoT Greengrass, consultethe section called "Cómo AWS IoT Greengrass funciona con IAM".

Público

La forma de usar AWS Identity and Access Management (IAM) varía según el trabajo en el que se realice AWS IoT Greengrass.

Usuario del servicio: si utiliza el AWS IoT Greengrass servicio para realizar su trabajo, el administrador le proporcionará las credenciales y los permisos que necesita. A medida que vaya utilizando más AWS IoT Greengrass funciones para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una característica en AWS IoT Greengrass, consulte Solución de problemas de administración de identidades y accesos en AWS IoT Greengrass.

Administrador de servicios: si estás a cargo de AWS IoT Greengrass los recursos de tu empresa, probablemente tengas acceso total a ellos AWS IoT Greengrass. Su trabajo consiste en determinar a qué AWS IoT Greengrass funciones y recursos deben acceder los usuarios del servicio. A continuación, debe enviar solicitudes a su IAM administrador para cambiar los permisos de los usuarios del servicio. Revise la información de esta página para comprender los conceptos básicos deIAM. Para obtener más información sobre cómo su empresa puede utilizar IAM con AWS IoT Greengrass, consulteCómo AWS IoT Greengrass funciona con IAM.

IAMadministrador: si es IAM administrador, puede que desee obtener más información sobre cómo puede redactar políticas para administrar el acceso a ellas AWS IoT Greengrass. Para ver ejemplos de políticas AWS IoT Greengrass basadas en la identidad que puede utilizarIAM, consulte. Ejemplos de políticas basadas en identidades de AWS IoT Greengrass

Autenticación con identidades

La autenticación es la forma de iniciar sesión AWS con sus credenciales de identidad. Debe estar autenticado (con quien haya iniciado sesión AWS) como IAM usuario o asumiendo un IAM rol. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (IAMIdentity Center), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, el administrador configuró previamente la federación de identidades mediante roles. IAM Cuando accede AWS mediante la federación, asume indirectamente un rol.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte Cómo iniciar sesión Cuenta de AWS en su Guía del AWS Sign-In usuario.

Si accede AWS mediante programación, AWS incluye un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener más información sobre cómo usar el método recomendado para firmar las solicitudes usted mismo, consulte Firmar AWS API las solicitudes en la Guía del IAM usuario.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactorial (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactorial en la Guía del AWS IAM Identity Center usuario y Uso de la autenticación multifactorial (MFA) AWS en la Guía del IAM usuario.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos Servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de tareas que requieren que inicie sesión como usuario root, consulte Tareas que requieren credenciales de usuario root en la Guía del IAM usuario.

Usuarios y grupos de IAM

Un IAMusuario es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos utilizar credenciales temporales en lugar de crear IAM usuarios con credenciales de larga duración, como contraseñas y claves de acceso. Sin embargo, si tiene casos de uso específicos que requieren credenciales a largo plazo con IAM los usuarios, le recomendamos que rote las claves de acceso. Para obtener más información, consulte Rotar las claves de acceso con regularidad para los casos de uso que requieran credenciales de larga duración en la Guía del IAM usuario.

Un IAMgrupo es una identidad que especifica un conjunto de IAM usuarios. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos para grandes conjuntos de usuarios. Por ejemplo, puede asignar un nombre a un grupo IAMAdminsy concederle permisos para administrar IAM los recursos.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales de larga duración permanentes; no obstante, los roles proporcionan credenciales temporales. Para obtener más información, consulte Cuándo crear un IAM usuario (en lugar de un rol) en la Guía del IAM usuario.

IAMroles

Un IAMrol es una identidad dentro de ti Cuenta de AWS que tiene permisos específicos. Es similar a un IAM usuario, pero no está asociado a una persona específica. Puede asumir temporalmente un IAM rol en el AWS Management Console cambiando de rol. Puede asumir un rol llamando a una AWS API operación AWS CLI o o utilizando una operación personalizadaURL. Para obtener más información sobre los métodos de uso de roles, consulte Uso de IAM roles en la Guía del IAM usuario.

IAMlos roles con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información sobre los roles para la federación, consulte Creación de un rol para un proveedor de identidad externo en la Guía del IAM usuario. Si usa IAM Identity Center, configura un conjunto de permisos. Para controlar a qué pueden acceder sus identidades después de autenticarse, IAM Identity Center correlaciona el

conjunto de permisos con un rol en. IAM Para obtener información acerca de los conjuntos de permisos, consulte Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center .

- Permisos IAM de usuario temporales: un IAM usuario o rol puede asumir un IAM rol para asumir temporalmente diferentes permisos para una tarea específica.
- Acceso multicuenta: puedes usar un IAM rol para permitir que alguien (un responsable de confianza) de una cuenta diferente acceda a los recursos de tu cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunos Servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para conocer la diferencia entre las funciones y las políticas basadas en recursos para el acceso multicuenta, consulta el tema sobre el acceso a los recursos entre cuentas IAM en la Guía del IAM usuario.
- Acceso entre servicios: algunos Servicios de AWS utilizan funciones en otros. Servicios de AWS Por ejemplo, cuando realizas una llamada en un servicio, es habitual que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.
  - Sesiones de acceso directo (FAS): cuando utilizas un IAM usuario o un rol para realizar acciones en AWS ellas, se te considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FASutiliza los permisos del principal que llama a an Servicio de AWS, junto con los que solicitan, Servicio de AWS para realizar solicitudes a los servicios descendentes. FASlas solicitudes solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener detalles sobre la política a la hora de realizar FAS solicitudes, consulte Reenviar sesiones de acceso.
  - Función de servicio: una función de servicio es una IAMfunción que un servicio asume para realizar acciones en su nombre. Un IAM administrador puede crear, modificar y eliminar un rol de servicio desde dentroIAM. Para obtener más información, consulte Crear un rol para delegar permisos Servicio de AWS en un rol en el IAMManual del usuario.
  - Función vinculada a un servicio: una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un IAM administrador puede ver los permisos de los roles vinculados al servicio, pero no editarlos.
- Aplicaciones que se ejecutan en Amazon EC2: puedes usar un IAM rol para administrar las credenciales temporales de las aplicaciones que se ejecutan en una EC2 instancia y que realizan

AWS CLI o AWS API solicitan. Esto es preferible a almacenar las claves de acceso en la EC2 instancia. Para asignar un AWS rol a una EC2 instancia y ponerlo a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite que los programas que se ejecutan en la EC2 instancia obtengan credenciales temporales. Para obtener más información, consulte Uso de un IAM rol para conceder permisos a aplicaciones que se ejecutan en EC2 instancias de Amazon en la Guía del IAM usuario.

Para saber si se deben usar IAM roles o IAM usuarios, consulte Cuándo crear un IAM rol (en lugar de un usuario) en la Guía del IAM usuario.

Administración de acceso mediante políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan AWS como JSON documentos. Para obtener más información sobre la estructura y el contenido de los documentos de JSON políticas, consulte Descripción general de JSON las políticas en la Guía del IAM usuario.

Los administradores pueden usar AWS JSON las políticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Para conceder a los usuarios permiso para realizar acciones en los recursos que necesitan, un IAM administrador puede crear IAM políticas. A continuación, el administrador puede añadir las IAM políticas a las funciones y los usuarios pueden asumir las funciones.

IAMlas políticas definen los permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre el rol de AWS Management Console AWS CLI, el o el AWS API.

Políticas basadas en identidad

Las políticas basadas en la identidad son documentos de política de JSON permisos que se pueden adjuntar a una identidad, como un IAM usuario, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en

qué condiciones. Para obtener información sobre cómo crear una política basada en la identidad, consulte Creación de IAM políticas en la Guía del usuario. IAM

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y funciones de su empresa. Cuenta de AWS Las políticas administradas incluyen políticas AWS administradas y políticas administradas por el cliente. Para saber cómo elegir entre una política gestionada o una política integrada, consulte Elegir entre políticas gestionadas y políticas integradas en la Guía del IAM usuario.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de JSON política que se adjuntan a un recurso. Algunos ejemplos de políticas basadas en recursos son las políticas de confianza de IAM roles y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puede usar políticas AWS administradas desde una política IAM basada en recursos.

Listas de control de acceso () ACLs

Las listas de control de acceso (ACLs) controlan qué responsables (miembros de la cuenta, usuarios o roles) tienen permisos para acceder a un recurso. ACLsson similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de JSON políticas.

Amazon S3 AWS WAF y Amazon VPC son ejemplos de servicios compatiblesACLs. Para obtener más informaciónACLs, consulte la descripción general de la lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales y menos comunes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una función avanzada en la que se establecen los permisos máximos que una política basada en la identidad puede conceder a una IAM entidad (IAMusuario o rol). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte los límites de los permisos para IAM las entidades en la Guía del IAMusuario.
- Políticas de control de servicios (SCPs): SCPs son JSON políticas que especifican los permisos máximos para una organización o unidad organizativa (OU) AWS Organizations. AWS Organizations es un servicio para agrupar y administrar de forma centralizada varios de los Cuentas de AWS que son propiedad de su empresa. Si habilitas todas las funciones de una organización, puedes aplicar políticas de control de servicios (SCPs) a una o a todas tus cuentas. SCPLimita los permisos de las entidades en las cuentas de los miembros, incluidas las de cada una Usuario raíz de la cuenta de AWS. Para obtener más información sobre OrganizationsSCPs, consulte las políticas de control de servicios en la Guía del AWS Organizations usuario.
- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información, consulte las políticas de sesión en la Guía del IAM usuario.

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 IAM usuario.

Véase también

- the section called “Cómo AWS IoT Greengrass funciona con IAM”
- the section called “Ejemplos de políticas basadas en identidad”
- the section called “Solución de problemas de identidades y accesos”

Cómo AWS IoT Greengrass funciona con IAM

Antes de administrar el acceso a AWS IoT Greengrass, debe comprender las IAM funciones con las que puede utilizarlas AWS IoT Greengrass. IAM

IAMcaracterística	¿Compatible con Greengrass?
Políticas basadas en identidad con permisos de nivel de recursos	Sí
Políticas basadas en recursos	No
Listas de control de acceso (ACLs)	No
Autorización basada en etiquetas	Sí
Credenciales temporales	Sí
Roles vinculados al servicio	No
Roles de servicio	Sí

Para obtener una visión general de cómo funcionan otros AWS serviciosIAM, consulte AWS los servicios con los que funcionan IAM en la Guía del IAM usuario.

Políticas basadas en la identidad para AWS IoT Greengrass

Con las políticas IAM basadas en la identidad, puede especificar las acciones y los recursos permitidos o denegados y las condiciones en las que se permiten o deniegan las acciones. AWS IoT Greengrass admite claves de condiciones, recursos y acciones específicas. Para obtener información sobre todos los elementos que se utilizan en una política, consulte la referencia a los elementos de la IAM JSON política en la Guía del IAM usuario.

Acciones

Los administradores pueden usar AWS JSON políticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El `Action` elemento de una JSON política describe las acciones que puede utilizar para permitir o denegar el acceso en una política. Las acciones de política suelen tener el mismo nombre que

la AWS API operación asociada. Hay algunas excepciones, como las acciones que solo permiten permisos y que no tienen una operación coincidente. API También hay algunas operaciones que requieren varias acciones en una política. Estas acciones adicionales se denominan acciones dependientes.

Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Acciones políticas para AWS IoT Greengrass usar el `greengrass:` prefijo antes de la acción. Por ejemplo, para permitir que alguien utilice la `ListGroups` API operación para enumerar sus grupos Cuenta de AWS, debes incluir la `greengrass:ListGroups` acción en su política. Las instrucciones de política deben incluir un elemento `Action` o `NotAction`. AWS IoT Greengrass define su propio conjunto de acciones que describen las tareas que se pueden realizar con este servicio.

Para especificar varias acciones en una misma instrucción, inclúyalas entre corchetes (`[ ]`) y sepárelas por comas, tal y como se indica a continuación:

```
"Action": [
    "greengrass:action1",
    "greengrass:action2",
    "greengrass:action3"
]
```

Puede utilizar comodines (*) para especificar varias acciones. Por ejemplo, para especificar todas las acciones que comiencen con la palabra `List`, incluya la siguiente acción:

```
"Action": "greengrass:List*"
```

> **Note**
>
> Se recomienda evitar el uso de comodines para especificar todas las acciones disponibles para un servicio. Como práctica recomendada, debe conceder permisos de mínimo privilegio y acotar el alcance de los permisos en una política. Para obtener más información, consulte the section called “Conceda los mínimos permisos posibles”.

Para ver la lista completa de AWS IoT Greengrass acciones, consulte las acciones definidas por AWS IoT Greengrass en la Guía del IAM usuario.

Recursos

Los administradores pueden usar AWS JSON políticas 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` JSON de política especifica el objeto o los objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Como práctica recomendada, especifique un recurso mediante su nombre de recurso de Amazon (ARN). Puede hacerlo para acciones que admitan un tipo de recurso específico, conocido como permisos de nivel de recurso.

Para las acciones que no admiten permisos de nivel de recurso, como las operaciones de descripción, utilice un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

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

La siguiente tabla contiene el AWS IoT Greengrass recurso ARNs que se puede utilizar como `Resource` elemento de una declaración de política. Para ver un mapeo de los permisos a nivel de recursos admitidos para AWS IoT Greengrass las acciones, consulte las acciones definidas por AWS IoT Greengrass en la Guía del IAMusuario.

Recurso	ARN
Group	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}`
GroupVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/versions/${VersionId}`
CertificateAuthority	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/certificateauthorities/${CertificateAuthorityId}`
Deployment	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/deployments/${DeploymentId}`

Recurso	ARN
BulkDeployment	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/bulk/deployments/${BulkDeploymentId}
ConnectorDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}
ConnectorDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}/versions/${VersionId}
CoreDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}
CoreDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}/versions/${VersionId}
DeviceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}
DeviceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}/versions/${VersionId}
FunctionDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}
FunctionDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}/versions/${VersionId}
LoggerDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}

Recurso	ARN
LoggerDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}/versions/${VersionId}`
ResourceDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}`
ResourceDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}/versions/${VersionId}`
SubscriptionDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}`
SubscriptionDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}/versions/${VersionId}`
ConnectivityInfo	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/things/${ThingName}/connectivityInfo`

El siguiente `Resource` elemento de ejemplo especifica el ARN número de un grupo en la región EE.UU. Oeste (Oregón) en: Cuenta de AWS `123456789012`

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

O bien, para especificar todos los grupos que pertenecen a un Cuenta de AWS grupo específico Región de AWS, utilice el comodín en lugar del identificador del grupo:

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/*"
```

Algunas AWS IoT Greengrass acciones (por ejemplo, algunas operaciones de lista) no se pueden realizar en un recurso específico. En dichos casos, debe utilizar solo el carácter comodín.

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

Para especificar varios recursos ARNs en una sentencia, enumérelos entre corchetes (`[]`) y sepárelos con comas, de la siguiente manera:

```
"Resource": [
    "resource-arn1",
    "resource-arn2",
    "resource-arn3"
]
```

Para obtener más información sobre los ARN formatos, consulte Nombres de recursos de Amazon (ARNs) y espacios AWS de nombres de servicios en. Referencia general de Amazon Web Services

Claves de condición

Los administradores pueden usar AWS JSON políticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` (o bloque de `Condition`) permite especificar condiciones en las que entra en vigor una instrucción. El elemento `Condition` es opcional. Puede crear expresiones condicionales que utilicen operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud.

Si especifica varios elementos de `Condition` en una instrucción o varias claves en un único elemento de `Condition`, AWS las evalúa mediante una operación `AND` lógica. Si especifica varios valores para una única clave de condición, AWS evalúa la condición mediante una `OR` operación lógica. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puede utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puede conceder a un IAM usuario permiso para acceder a un recurso solo si está etiquetado con su nombre de IAM usuario. Para obtener más información, consulte los elementos de IAM política: variables y etiquetas en la Guía del IAM usuario.

AWS admite claves de condición globales y claves de condición específicas del servicio. Para ver todas las claves de condición AWS globales, consulte las claves de contexto de condición AWS globales en la Guía del IAMusuario.

AWS IoT Greengrass admite las siguientes claves de condición globales.

Clave	Descripción
`aws:CurrentTime`	Filtra el acceso al comprobar las condiciones de fecha y hora de la fecha y hora actual.
`aws:EpochTime`	Filtra el acceso al comprobar las condiciones de fecha y hora de la fecha y hora actual en formato de hora Unix.
`aws:MultiFactorAuthAge`	Filtra el acceso comprobando con cuánto tiempo (en segundos) se emitieron las credenciales de seguridad validadas mediante la autenticación multifactorial (MFA) en la solicitud. MFA
`aws:MultiFactorAuthPresent`	Filtra el acceso comprobando si se utilizó la autenticación multifactorial (MFA) para validar las credenciales de seguridad temporales que hicieron la solicitud actual.
`aws:RequestTag/${TagKey}`	Filtra las solicitudes de creación en función del conjunto de valores permitidos para cada una de las etiquetas obligatorias.
`aws:ResourceTag/${TagKey}`	Filtra acciones en función del valor de la etiqueta asociado con el recurso.
`aws:SecureTransport`	Filtra el acceso comprobando si la solicitud se envió utilizand oSSL.
`aws:TagKeys`	Filtra las solicitudes de creación en función de la presencia de etiquetas obligatorias en la solicitud.
`aws:UserAgent`	Filtra el acceso por aplicación cliente del solicitante.

Para obtener más información, consulte las claves de contexto de las condiciones AWS globales en la Guía del IAM usuario.

Ejemplos

Para ver ejemplos de políticas AWS IoT Greengrass basadas en la identidad, consulte. the section called “Ejemplos de políticas basadas en identidad”

Políticas basadas en recursos para AWS IoT Greengrass

AWS IoT Greengrass no admite políticas basadas en recursos.

Listas de control de acceso () ACLs

AWS IoT Greengrass no es compatible ACLs.

Autorización basada en etiquetas de AWS IoT Greengrass

Puedes adjuntar etiquetas a AWS IoT Greengrass los recursos compatibles o pasarles etiquetas en una solicitud AWS IoT Greengrass. Para controlar el acceso utilizando etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` o `aws:TagKeys`. Para obtener más información, consulte Etiquetado de los recursos de Greengrass.

IAMroles para AWS IoT Greengrass

Un IAMrol es una entidad dentro de usted Cuenta de AWS que tiene permisos específicos.

Utilizar credenciales temporales con AWS IoT Greengrass

Las credenciales temporales se utilizan para iniciar sesión con la federación, asumir un IAM rol o asumir un rol multicuenta. Las credenciales de seguridad temporales se obtienen llamando a AWS STS API operaciones como AssumeRoleo GetFederationToken.

En el núcleo de Greengrass, las credenciales temporales para el rol de grupo están disponibles para las funciones de Lambda y conectores definidos por el usuario. Si sus funciones Lambda utilizan AWS SDK, no necesita añadir lógica para obtener las credenciales, ya que AWS SDK lo hace por usted.

Roles vinculados al servicio

AWS IoT Greengrass no admite funciones vinculadas a servicios.

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 IAM cuenta y son propiedad de la cuenta. Esto significa que un IAM administrador puede cambiar los permisos de este rol. Sin embargo, hacerlo podría deteriorar la funcionalidad del servicio.

AWS IoT Greengrass utiliza un rol de servicio para acceder a algunos de sus AWS recursos en su nombre. Para obtener más información, consulte the section called “Rol de servicio de Greengrass”.

Elegir un IAM rol en la AWS IoT Greengrass consola

En la AWS IoT Greengrass consola, puede que tengas que elegir un rol de servicio de Greengrass o un rol de grupo de Greengrass de una lista de IAM roles de tu cuenta.

- El rol de servicio de Greengrass le permite acceder AWS IoT Greengrass a sus AWS recursos en otros servicios en su nombre. Normalmente, no es necesario elegir el rol de servicio porque la consola puede crearlo y configurarlo automáticamente. Para obtener más información, consulte the section called “Rol de servicio de Greengrass”.
- El rol de grupo de Greengrass se usa para permitir que las funciones y los conectores de Greengrass Lambda del grupo accedan a sus recursos. AWS También puede conceder AWS IoT Greengrass permisos para exportar flujos a AWS servicios y escribir registros. CloudWatch Para obtener más información, consulte the section called “Rol de grupo de Greengrass”.

Rol de servicio de Greengrass

El rol de servicio de Greengrass es un rol de servicio de AWS Identity and Access Management (IAM) que autoriza a AWS IoT Greengrass a acceder a recursos de servicios de AWS en su nombre. Esto permite a AWS IoT Greengrass realizar tareas esenciales como, por ejemplo, recuperar funciones de AWS Lambda y administrar sombras de AWS IoT.

Para permitir a AWS IoT Greengrass acceder a sus recursos, el rol de servicio de Greengrass debe estar asociado a su Cuenta de AWS y especificar AWS IoT Greengrass como entidad de confianza. El rol debe incluir la política administrada AWSGreenGrassResourceAccessRolePolicy o una política personalizada que defina permisos equivalentes para las características de AWS IoT Greengrass que utilice. AWS mantiene esta política y define el conjunto de permisos que AWS IoT Greengrass utiliza para acceder a los recursos de AWS.

Puede reutilizar el mismo rol de servicio de Greengrass en todas las Región de AWS, pero lo debe asociar a su cuenta en cada Región de AWS donde utilice AWS IoT Greengrass. La implementación de grupos falla si el rol de servicio no existe en la Cuenta de AWS y región actual.

En las secciones siguientes se describe cómo crear y administrar el rol de servicio de Greengrass en la AWS Management Console o la AWS CLI.

- Administración del rol de servicio (consola)

- Administración del rol de servicio (CLI)

> **Note**
>
> Además del rol de servicio que autoriza el acceso de nivel de servicio, puede asignar un rol de grupo a un AWS IoT Greengrass. El rol de grupo es un rol independiente de IAM que controla cómo las funciones de Lambda Greengrass y los conectores del grupo pueden acceder a los servicios de AWS.

Administración del rol de servicio de Greengrass (consola)

La consola de AWS IoT facilita la administración del rol de servicio de Greengrass. Por ejemplo, al crear o implementar un grupo de Greengrass, la consola comprueba si su Cuenta de AWS está asociada a un rol de servicio de Greengrass en la Región de AWS seleccionada actualmente en la consola. De lo contrario, la consola puede crear y configurar un rol de servicio por usted. Para obtener más información, consulte the section called “Creación del rol de servicio de Greengrass”.

Puede utilizar la consola AWS IoT para las siguientes tareas de administración de roles:

- Buscar el rol de servicio de Greengrass
- Creación del rol de servicio de Greengrass
- Cambiar el rol de servicio de Greengrass
- Desasociar el rol de servicio de Greengrass

> **Note**
>
> El usuario que ha iniciado sesión en la consola debe tener permisos para ver, crear o cambiar el rol de servicio.

Buscar el rol de servicio de Greengrass (consola)

Siga estos pasos para buscar el rol de servicio que AWS IoT Greengrass utiliza en la Región de AWS actual.

1. En el panel de navegación de la consola AWS IoT, seleccione Configuración.
2. Desplácese hasta la sección Greengrass service role (Rol de servicio de Greengrass) para ver el rol de servicio y sus políticas.

   Si no ve ningún rol de servicio, puede dejar que la consola cree o configure uno por usted. Para obtener más información, consulte Creación del rol de servicio de Greengrass.

Creación del rol de servicio de Greengrass (consola)

La consola puede crear y configurar un rol de servicio de Greengrass predeterminado por usted. Este rol incluye las siguientes propiedades.

Propiedad	Valor
Nombre	`Greengrass_ServiceRole`
Entidad de confianza	`AWS service: greengrass`
Política	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> Si la configuración del dispositivo de Greengrass crea el rol de servicio, el nombre del rol es `GreengrassServiceRole_`*`random-string`*.

Al crear o implementar un grupo de Greengrass desde la consola AWS IoT, la consola comprueba si hay un rol de servicio de Greengrass asociado a su Cuenta de AWS en la Región de AWS seleccionada actualmente en la consola. Si no lo hay, la consola le pedirá que permita a AWS IoT Greengrass leer y escribir en los servicios de AWS en su nombre.

Si concede permiso, la consola comprueba si existe un rol denominado `Greengrass_ServiceRole` en su Cuenta de AWS.

- Si el rol existe, la consola asocia el rol de servicio a su Cuenta de AWS en la Región de AWS actual.

- Si el rol no existe, la consola crea un rol de servicio de Greengrass predeterminado y lo asocia a su Cuenta de AWS en la Región de AWS actual.

> **Note**
>
> Si desea crear un rol de servicio con políticas de rol personalizadas, utilice la consola de IAM para crear o modificar el rol. Para obtener más información, consulte Creación de un rol para delegar permisos a un servicio AWS o Modificación de un rol en la Guía del usuario de IAM. Asegúrese de que el rol concede permisos equivalentes a la política administrada de `AWSGreengrassResourceAccessRolePolicy` para las características y recursos que utiliza. Le recomendamos que incluya también las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en su política de confianza para ayudar a prevenir el problema de seguridad del suplente confuso. Las claves de contexto de condición restringen el acceso para permitir solo las solicitudes que provienen de la cuenta especificada y del espacio de trabajo de Greengrass. Para obtener más información sobre el problema del suplente confuso, consulte Prevención del suplente confuso entre servicios.
> Si crea un rol de servicio, vuelva a la consola AWS IoT y asocie el rol al grupo. Puede hacerlo en el rol de servicio de Greengrass en la página Configuración del grupo.

Cambiar el rol de servicio de Greengrass (consola)

Utilice el siguiente procedimiento para seleccionar un rol de servicio de Greengrass diferente y asociarlo a su Cuenta de AWS en la Región de AWS seleccionada actualmente en la consola.

1. En el panel de navegación de la consola AWS IoT, seleccione Configuración.
2. En Rol de servicio de Greengrass, seleccione Elegir un rol diferente.

   Se abre el cuadro de diálogo Actualizar el rol de servicio de Greengrass y muestra los roles de IAM Cuenta de AWS que se definen AWS IoT Greengrass como una entidad de confianza.
3. Elija el rol de servicio de Greengrass que desee asignar.
4. Elija Adjuntar rol.

> **Note**
>
> Para permitir que la consola cree un rol de servicio de Greengrass predeterminado por usted, seleccione Create role for me (Crear rol por mí) en lugar de seleccionar un rol de la lista. El enlace Crear rol por mí no aparece si hay un rol denominado `Greengrass_ServiceRole` en su Cuenta de AWS.

Desasociar el rol de servicio de Greengrass (consola)

Utilice el siguiente procedimiento para desasociar el rol de servicio de Greengrass de su Cuenta de AWS en la Región de AWS seleccionada actualmente en la consola. Este revoca los permisos de AWS IoT Greengrass para acceder a los servicios de AWS en la Región de AWS actual.

> **Important**
>
> La desasociación del rol de servicio podría interrumpir las operaciones activas.

1. En el panel de navegación de la consola AWS IoT, seleccione Configuración.
2. En Rol de servicio de Greengras, seleccione Desasociar rol.
3. En el cuadro de diálogo de confirmación, elija Desconectar.

> **Note**
>
> Si ya no necesita el rol, puede eliminarlo en la consola de IAM. Para obtener más información, consulte Eliminación de roles o perfiles de instancia en la Guía del usuario de IAM.
> Otros roles podrían permitir que AWS IoT Greengrass obtenga acceso a los recursos. Para buscar todos los roles que permiten que AWS IoT Greengrass asuma los permisos en su nombre, en la consola de IAM, en la página Roles, busque los roles que incluyan AWS service: greengrass en la columna Entidades de confianza.

Administración del rol de servicio de Greengrass (CLI)

En los procedimientos siguientes, suponemos que la AWS CLI está instalada y configurada para utilizar su ID de Cuenta de AWS. Para obtener más información, consulte Instalación de la interfaz de línea de comandos AWS y Configuración de la AWS CLIen la AWS Command Line InterfaceGuía del usuario.

Puede utilizar la AWS CLI para las siguientes tareas de administración de roles:

- Obtener el rol de servicio de Greengrass
- Creación del rol de servicio de Greengrass
- Eliminar el rol de servicio de Greengrass

Obtener el rol de servicio de Greengrass (CLI)

Utilice el procedimiento siguiente para descubrir si un rol de servicio de Greengrass está asociado a su Cuenta de AWS en una Región de AWS.

- Obtenga el rol de servicio. Sustituya *región* por su Región de AWS (por ejemplo, `us-west-2`).

  ```
  aws Greengrass get-service-role-for-account --region region
  ```

  Si ya hay un rol de servicio de Greengrass asociado a su cuenta, se devuelven los siguientes metadatos de rol.

  ```
  {
    "AssociatedAt": "timestamp",
    "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  Si no se devuelve ningún metadato de rol, entonces debe crear el rol de servicio (si no existe) y asociarlo a su cuenta en la Región de AWS.

Creación del rol de servicio de Greengrass (CLI)

Siga los pasos que se indican a continuación para crear un rol y asociarlo a su Cuenta de AWS.

Para crear el rol de servicio mediante IAM

1. Cree el rol con una política de confianza que permita a AWS IoT Greengrass adoptar el rol. Este ejemplo crea un rol denominado `Greengrass_ServiceRole`, pero puede utilizar un nombre distinto. Le recomendamos que incluya también las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en su política de confianza para ayudar a prevenir el problema de seguridad del suplente confuso. Las claves de contexto de condición restringen el acceso para permitir solo las solicitudes que provienen de la cuenta especificada y del espacio de trabajo de Greengrass. Para obtener más información sobre el problema del suplente confuso, consulte Prevención del suplente confuso entre servicios.

   Linux, macOS, or Unix

   ```
   aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
   document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "greengrass.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
           "StringEquals": {
             "aws:SourceAccount": "account-id"
           },
           "ArnLike": {
             "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
           }
         }
       }
     ]
   }'
   ```

Windows command prompt

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-
policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
\"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
\":\"arn:aws:greengrass:region:account-id:*\"},\"StringEquals\":
{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

2. Copie el ARN del rol de los metadatos del rol en la salida. Puede utilizar el ARN para asociar el rol a su cuenta.
3. Asocie la política de `AWSGreengrassResourceAccessRolePolicy` al rol.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

Para asociar el rol de servicio con su Cuenta de AWS

- Asocie el rol a su cuenta. Reemplace *`arn-rol`* por el ARN del rol de servicio y *`región`* por su Región de AWS (por ejemplo, `us-west-2`).

```
aws greengrass associate-service-role-to-account --role-arn role-arn --
region region
```

Si se ejecuta correctamente, se devuelve la siguiente respuesta.

```
{
  "AssociatedAt": "timestamp"
}
```

Eliminar el rol de servicio de Greengrass (CLI)

Utilice los pasos siguientes para desasociar el rol de servicio de Greengrass de su Cuenta de AWS.

- Desasocie el rol de servicio de su cuenta. Sustituya *`región`* por su Región de AWS (por ejemplo, `us-west-2`).

```
aws greengrass disassociate-service-role-from-account --region region
```

Si se ejecuta correctamente, se devuelve la siguiente respuesta.

```
{
  "DisassociatedAt": "timestamp"
}
```

> **Note**
>
> Debe eliminar el rol de servicio si no lo está utilizando en ninguna Región de AWS. Use primero delete-role-policy para desasociar la política administrada `AWSGreengrassResourceAccessRolePolicy` del rol y, a continuación, utilice delete-role para eliminar el rol. Para obtener más información, consulte Eliminación de roles o perfiles de instancia en la Guía del usuario de IAM.

Véase también

- Creación de un rol para delegar permisos a un servicio AWSen la Guía del usuario de IAM
- Modificación de un rol en la Guía del usuario de IAM
- Eliminación de roles o perfiles de instancia en la Guía del usuario de IAM
- comandos de AWS IoT Greengrass en la Referencia de los comandos de AWS CLI
  - associate-service-role-to-account
  - disassociate-service-role-from-account
  - get-service-role-for-account
- Comandos de IAM en la Referencia de los comandos de AWS CLI
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

Rol de grupo de Greengrass

El rol de grupo de Greengrass es un rol de IAM que autoriza la ejecución de código en un núcleo de Greengrass para acceder a sus recursos de AWS. Cree el rol y administre los permisos en AWS Identity and Access Management (IAM) y asocie el rol a su grupo de Greengrass. Un grupo de Greengrass tiene un rol de grupo. Para agregar o cambiar permisos, puede asociar un rol diferente o cambiar las políticas de IAM asociadas al rol.

El rol debe definir AWS IoT Greengrass como entidad de confianza. En función de su caso comercial, el rol de grupo podría contener políticas de IAM que definan:

- Permisos para que las funciones de Lambda definidas por el usuario accedan a servicios de AWS.
- Permisos para que los conectores accedan a los servicios de AWS.
- Permisos para que el administrador de secuencias exporte transmisiones a AWS IoT Analytics y flujos de datos de Kinesis.
- Permisos para permitir registros de CloudWatch.

En las secciones siguientes se describe cómo asociar o desasociar un rol de grupo de Greengrass en la AWS Management Console o en la AWS CLI.

- Administración del rol de grupo (consola)
- Administración del rol de grupo (CLI)

> **Note**
>
> Además del rol de grupo que autoriza el acceso desde el núcleo de Greengrass, puede asignar un rol de servicio de Greengrass que permita a AWS IoT Greengrass acceder a los recursos de AWS en su nombre.

Administración del rol de grupo de Greengrass (consola)

Puede utilizar la consola AWS IoT para las siguientes tareas de administración de roles:

- Buscar el rol de grupo de Greengrass
- Agregar o cambiar el rol de grupo de Greengrass

- Quitar el rol de grupo Greengrass

> **Note**
>
> El usuario que ha iniciado sesión en la consola debe tener permisos para administrar el rol.

Buscar el rol de grupo de Greengrass (consola)

Siga estos pasos para buscar el rol que está asociado a un grupo de Greengrass.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. En la página de configuración de grupo, elija Ver configuración.

Si un rol está asociado al grupo, aparece en Rol de grupo.

Agregar o cambiar el rol de grupo de Greengrass (consola)

Siga estos pasos para elegir un rol de IAM de su Cuenta de AWS para agregarlo a un grupo de Greengrass.

Un rol de grupo tiene los siguientes requisitos:

- AWS IoT Greengrass definido como entidad de confianza.
- Las políticas de permisos asociadas al rol deben conceder los permisos a los recursos de AWS necesarios para las funciones de Lambda y conectores del grupo y para los componentes del sistema de Greengrass.

> **Note**
>
> Le recomendamos que incluya también las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en su política de confianza para ayudar

a prevenir el problema de seguridad del suplente confuso. Las claves de contexto de condición restringen el acceso para permitir solo las solicitudes que provienen de la cuenta especificada y del espacio de trabajo de Greengrass. Para obtener más información sobre el problema del suplente confuso, consulte Prevención del suplente confuso entre servicios.

Utilice la consola de IAM para crear y configurar el rol y sus permisos. Para ver los pasos que crean un rol de ejemplo que permite el acceso a una tabla de Amazon DynamoDB, consulte the section called “Configuración del rol del grupo”. Para ver los pasos generales, consulte Creación de un rol para un servicio de AWS (consola) en la Guía del usuario de IAM.

Una vez configurado el rol, utilice la consola AWS IoT para agregar el rol al grupo.

> **Note**
>
> Este procedimiento solo es necesario para elegir un rol para el grupo. No es necesario después de cambiar los permisos del rol de grupo seleccionado actualmente.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. En la página de configuración de grupo, elija Ver configuración.
4. En Rol de grupo, elija agregar o cambiar el rol:
   - Para añadir el rol, elija Asociar rol y, a continuación, seleccione el suyo de la lista de roles. Estos son los roles de su Cuenta de AWS que definen AWS IoT Greengrass como entidad de confianza.
   - Para elegir un rol diferente, elija Editar rol y luego seleccione su rol de la lista de roles.
5. Seleccione Save.

Quitar el rol de grupo de Greengrass (consola)

Siga estos pasos para desasociar el rol de un grupo Greengrass.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. En la página de configuración de grupo, elija Ver configuración.
4. En Rol del grupo, elija Desasociar rol.
5. En el cuadro de diálogo de confirmación, elija Desasociar rol. Este paso quita el rol del grupo, pero no lo elimina. Si desea eliminar el rol, utilice la consola de IAM.

Administración del rol de grupo de Greengrass (CLI)

Puede utilizar la AWS CLI para las siguientes tareas de administración de roles:

- Obtener el rol de grupo de Greengrass
- Crear el rol de grupo de Greengrass
- Quitar el rol de grupo Greengrass

Obtener el rol de grupo de Greengrass (CLI)

Siga estos pasos para averiguar si un grupo Greengrass tiene un rol asociado.

1. Obtenga el ID del grupo de destino de la lista de sus grupos.

   ```
   aws greengrass list-groups
   ```

   A continuación se muestra un ejemplo de respuesta de `list-groups`. Cada grupo de la respuesta incluye una propiedad `Id` que contiene el ID de grupo.

   ```
   {
       "Groups": [
           {
               "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
   groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
   a50e-7d356EXAMPLE",
               "Name": "MyFirstGroup",
               "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
               "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
   ```

```
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

Para obtener más información, incluidos ejemplos que utilizan la opción `query` para filtrar resultados, consulte the section called “Obtener el ID del grupo”.

2. Copie el `Id` del grupo de destino de la salida.
3. Obtener el rol de grupo. Sustituya *`ID-grupo`* por el ID del grupo de destino.

```
aws greengrass get-associated-role --group-id group-id
```

Si un rol está asociado con el grupo Greengrass, se devuelven los siguientes metadatos de rol.

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

Si el grupo no tiene un rol asociado, se devuelve el siguiente error.

```
An error occurred (404) when calling the GetAssociatedRole operation: You need to
 attach an IAM role to this deployment group.
```

Crear el rol de grupo de Greengrass (CLI)

Siga estos pasos para crear un rol y asociarlo a un grupo de Greengrass.

Para crear el rol de grupo mediante IAM

1. Cree el rol con una política de confianza que permita a AWS IoT Greengrass adoptar el rol. Este ejemplo crea un rol denominado `MyGreengrassGroupRole`, pero puede utilizar un nombre distinto. Le recomendamos que incluya también las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en su política de confianza para ayudar a prevenir el problema de seguridad del suplente confuso. Las claves de contexto de condición restringen el acceso para permitir solo las solicitudes que provienen de la cuenta especificada y del espacio de trabajo de Greengrass. Para obtener más información sobre el problema del suplente confuso, consulte Prevención del suplente confuso entre servicios.

   Linux, macOS, or Unix

   ```
   aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-
   document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "greengrass.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
           "StringEquals": {
             "aws:SourceAccount": "account-id"
           },
           "ArnLike": {
             "aws:SourceArn": "arn:aws:greengrass:region:account-id:/greengrass/
   groups/group-id"
           }
         }
       }
     ]
   }'
   ```

Windows command prompt

```
aws iam create-role --role-name MyGreengrassGroupRole --assume-role-
policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
\"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
\":\"arn:aws:greengrass:region:account-id:/greengrass/groups/group-id\"},
\"StringEquals\":{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

2. Copie el ARN del rol de los metadatos del rol en la salida. Puede utilizar el ARN para asociar el rol con su grupo.
3. Asocie políticas administradas o en línea al rol para dar soporte a su caso de negocio. Por ejemplo, si una función de Lambda definida por el usuario lee desde Amazon S3, puede asociar la política de `AmazonS3ReadOnlyAccess` administrada al rol.

```
aws iam attach-role-policy --role-name MyGreengrassGroupRole --policy-arn
 arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

Si se realiza correctamente, no se devuelve ninguna respuesta.

Para asociar el rol con su grupo de Greengrass

1. Obtenga el ID del grupo de destino de la lista de sus grupos.

```
aws greengrass list-groups
```

A continuación se muestra un ejemplo de respuesta de `list-groups`. Cada grupo de la respuesta incluye una propiedad `Id` que contiene el ID de grupo.

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
```

```
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

Para obtener más información, incluidos ejemplos que utilizan la opción `query` para filtrar resultados, consulte the section called “Obtener el ID del grupo”.

2. Copie el `Id` del grupo de destino de la salida.
3. Asocie el rol a su grupo. Sustituya *`id-grupo`* por el ID del grupo de destino y *`arn-rol`* por el ARN del rol de grupo.

```
aws greengrass associate-role-to-group --group-id group-id --role-arn role-arn
```

Si se ejecuta correctamente, se devuelve la siguiente respuesta.

```
{
  "AssociatedAt": "timestamp"
}
```

Quitar el rol de grupo Greengrass (CLI)

Siga estos pasos para desasociar el rol de grupo del grupo de Greengrass.

1. Obtenga el ID del grupo de destino de la lista de sus grupos.

```
aws greengrass list-groups
```

A continuación se muestra un ejemplo de respuesta de `list-groups`. Cada grupo de la respuesta incluye una propiedad `Id` que contiene el ID de grupo.

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

Para obtener más información, incluidos ejemplos que utilizan la opción `query` para filtrar resultados, consulte the section called “Obtener el ID del grupo”.

2. Copie el `Id` del grupo de destino de la salida.
3. Desasociar el rol del grupo. Sustituya *`ID-grupo`* por el ID del grupo de destino.

```
aws greengrass disassociate-role-from-group --group-id group-id
```

Si se ejecuta correctamente, se devuelve la siguiente respuesta.

```
{
  "DisassociatedAt": "timestamp"
}
```

> **Note**
>
> Puede eliminar el rol de grupo si no lo está utilizando. Use primero delete-role-policy para desasociar la política administrada del rol y, a continuación, utilice delete-role para eliminar el rol. Para obtener más información, consulte Eliminación de roles o perfiles de instancia en la Guía del usuario de IAM.

Véase también

- Temas relacionados en la Guía del usuario de IAM
  - Creación de un rol para delegar permisos a un servicio de AWS
  - Modificación de un rol
  - Adición y eliminación de permisos de identidad de IAM
  - Eliminación de roles o perfiles de instancia
- comandos de AWS IoT Greengrass en la Referencia de los comandos de AWS CLI
  - grupos de lista
  - associate-role-to-group
  - disassociate-role-from-group
  - get-associated-role
- Comandos de IAM en la Referencia de los comandos de AWS CLI

- attach-role-policy
- create-role
- delete-role
- delete-role-policy

Prevención del suplente confuso entre servicios

El problema de la sustitución confusa es una cuestión de seguridad en la que una entidad que no tiene permiso para realizar una acción puede obligar a una entidad con más privilegios a realizar la acción. En AWS, la suplantación entre servicios puede dar lugar al problema del suplente confuso. La suplantación entre servicios puede producirse cuando un servicio (el servicio que lleva a cabo las llamadas) llama a otro servicio (el servicio al que se llama). El servicio que lleva a cabo las llamadas se puede manipular para utilizar sus permisos a fin de actuar en función de los recursos de otro cliente de una manera en la que no debe tener permiso para acceder. Para evitarlo, AWS proporciona herramientas que lo ayudan a proteger sus datos para todos los servicios con entidades principales de servicio a las que se les ha dado acceso a los recursos de su cuenta.

Se recomienda utilizar las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en las políticas de recursos para limitar los permisos que AWS IoT Greengrass concede a otro servicio para el recurso. Si se utilizan ambas claves de contexto de condición global, el valor `aws:SourceAccount` y la cuenta del valor `aws:SourceArn` deben utilizar el mismo ID de cuenta cuando se utilicen en la misma declaración de política.

El valor de `aws:SourceArn` debe ser el recurso de cliente de Greengrass asociado a la solicitud `sts:AssumeRole`.

La forma más eficaz de protegerse contra el problema del suplente confuso es utilizar la clave de contexto de condición global de `aws:SourceArn` con el ARN completo del recurso. Si no conoce el ARN completo del recurso o si especifica varios recursos, utilice la clave de condición de contexto global `aws:SourceArn` con comodines (`*`) para las partes desconocidas del ARN. Por ejemplo, `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:*`.

Para ver ejemplos de las políticas que utilizan las claves de contexto de condición global de `aws:SourceArn` y `aws:SourceAccount`, consulte los siguientes temas:

- Creación del rol de servicio de Greengrass
- Crear el rol de grupo de Greengrass

- Cree y configure un rol de ejecución IAM para implementaciones masivas

Ejemplos de políticas basadas en identidades de AWS IoT Greengrass

De forma predeterminada, los usuarios y los roles de IAM no tienen permiso para crear, ver ni modificar recursos de AWS IoT Greengrass. Tampoco pueden realizar tareas mediante la AWS Management Console, la AWS CLI, o la API de AWS. Un administrador de IAM debe crear políticas de IAM que concedan permisos a los usuarios y a los roles para realizar operaciones de la API concretas en los recursos especificados que necesiten. El administrador debe adjuntar esas políticas a los usuarios o grupos de IAM que necesiten esos permisos.

Prácticas recomendadas relativas a políticas

Las políticas basadas en identidades determinan si alguien puede crear, acceder o eliminar los recursos de AWS IoT Greengrass 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 identidad:

- Comience con las políticas administradas por AWS y continúe con los permisos de privilegio mínimo: a fin de comenzar a conceder permisos a los usuarios y las cargas de trabajo, utilice las políticas administradas por AWS, que conceden permisos para muchos casos de uso comunes. Están disponibles en la Cuenta de AWS. Se recomienda definir políticas administradas por el cliente de AWS específicas para sus casos de uso a fin de reducir aún más los permisos. Con el fin de obtener más información, consulte las políticas administradas por AWS o las políticas administradas por AWS para funciones de trabajo en la Guía 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 de usuario de IAM.
- Use 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 política para especificar que todas las solicitudes deben enviarse utilizando SSL. También puede usar condiciones para conceder acceso a acciones de servicios si se emplean a través de un Servicio de AWS determinado, como por ejemplo AWS CloudFormation. Para obtener más información, consulte Elementos de la política JSON de IAM: condición en la Guía del usuario de IAM.

- Use 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. IAM Access Analyzer proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para obtener más información, consulte la política de validación del Analizador de acceso de IAM en la Guía de usuario de IAM.
- Solicite la autenticación multifactor (MFA): si se encuentra en una situación en la que necesita usuarios raíz o de IAM en su Cuenta de AWS, active la MFA para mayor seguridad. Para solicitar la MFA cuando se invocan las operaciones de la API, agregue las condiciones de MFA a sus políticas. Para obtener más información, consulte Configuración de acceso a una API protegida por MFA en la Guía de usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte las Prácticas recomendadas de seguridad en IAM en la Guía de usuario de IAM.

Políticas administradas de AWS para AWS IoT Greengrass

AWS IoT Greengrass mantiene las siguientes políticas administradas de AWS que puede utilizar para conceder permisos a roles y usuarios de IAM.

Política	Descripción
AWSGreengrassFullAccess	Permite todas las acciones de AWS IoT Greengrass para todos los recursos de AWS. Esta política se recomienda para los administradores de servicios de AWS IoT Greengrass o para realizar pruebas.
AWSGreengrassReadOnlyAccess	Permite las acciones `List` y `Get` de AWS IoT Greengrass para todos sus recursos de AWS.
AWSGreengrassResourceAccessRolePolicy	Permite el acceso a los recursos desde servicios de AWS, incluidos AWS Lambda y sombra de dispositivo de AWS IoT. Esta es la política predeterminada utilizada para el rol de servicio de Greengrass. Esta política está

Política	Descripción
	diseñada para proporcionar facilidad general de acceso. Puede definir una política personali zada que sea más restrictiva.
GreengrassOTAUpdateArtifactAccess	Permite el acceso de solo lectura a artefactos de actualización por vía inalámbrica (OTA) para el software AWS IoT Greengrass Core en todas las regiones de Región de AWS.

Ejemplos de políticas

En el ejemplo siguiente, las políticas definidas por el cliente conceden permisos para situaciones comunes.

Ejemplos

- Permitir a los usuarios consultar sus propios permisos

Para obtener más información acerca de cómo crear una política basada en identidad de IAM con estos documentos de políticas de JSON de ejemplo, consulte Creación de políticas en la pestaña JSON en la Guía del usuario de IAM.

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 adjuntan a la identidad de sus usuarios. Esta política incluye permisos para llevar a cabo esta acción en la consola o mediante programación con la AWS CLI o la API de AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
```

```
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Solución de problemas de administración de identidades y accesos en AWS IoT Greengrass

Utilice la siguiente información como ayuda para diagnosticar y solucionar problemas comunes que pueden surgir al trabajar con un AWS IoT Greengrass IAM.

Problemas

- No estoy autorizado a realizar ninguna acción en AWS IoT Greengrass
- Error: Greengrass is not authorized to assume the Service Role associated with this account, o el error: Failed: TES service role is not associated with this account.
- Error: Permission denied when attempting to use role arn:aws:iam::<account-id>:role/<role-name> to access s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz.
- La sombra del dispositivo no se sincroniza con la nube.
- No estoy autorizado a realizar las siguientes tareas: PassRole

- Soy administrador y quiero permitir el acceso de otras personas AWS IoT Greengrass
- Quiero permitir que personas ajenas a mí accedan a mis recursos Cuenta de AWSAWS IoT Greengrass

Para obtener ayuda general de solución de problemas, consulte Solución de problemas.

No estoy autorizado a realizar ninguna acción en AWS IoT Greengrass

Si recibe un error que indica que no está autorizado para llevar a cabo una acción, debe ponerse en contacto con su administrador para recibir ayuda. Su administrador es la persona que le facilitó su nombre de usuario y contraseña.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM de `mateojackson` intenta ver detalles sobre una versión de definición del núcleo, pero no tiene permisos `greengrass:GetCoreDefinitionVersion`.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 greengrass:GetCoreDefinitionVersion on resource: resource: arn:aws:greengrass:us-
west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/
versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE
```

En este caso, Mateo pide a su administrador que actualice sus políticas de forma que pueda obtener acceso al recurso `arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE` mediante la acción `greengrass:GetCoreDefinitionVersion`.

Error: Greengrass is not authorized to assume the Service Role associated with this account, o el error: Failed: TES service role is not associated with this account.

Solución: Es posible que vea este error cuando la implementación no se realiza correctamente. Compruebe que un rol de servicio de Greengrass esté asociado a su Cuenta de AWS en la Región de AWS actual. Para obtener más información, consulte the section called “Administración del rol de servicio (CLI)” o the section called “Administración del rol de servicio (consola)”.

Error: Permission denied when attempting to use role arn:aws:iam::<account-id>:role/<role-name> to access s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz.

Solución: es posible que aparezca este error cuando se produce un error en una actualización over-the-air (OTA). En la política de rol de firma, añada la Región de AWS de destino como `Resource`. Esta función de firmante se utiliza para prefirmar la URL de S3 para la actualización de AWS IoT Greengrass software. Para obtener más información, consulte Rol de firmante de URL de S3.

La sombra del dispositivo no se sincroniza con la nube.

Solución: asegúrese de que AWS IoT Greengrass tiene permisos `iot:UpdateThingShadow` y `iot:GetThingShadow` acciones para el rol de servicio de Greengrass. Si el rol de servicio utiliza la política administrada `AWSGreengrassResourceAccessRolePolicy`, estos permisos se incluyen de forma predeterminada.

Consulte Solución de problemas con los tiempos de espera de la sincronización de sombras.

A continuación se indican problemas de IAM generales que puede encontrar al trabajar con AWS IoT Greengrass.

No estoy autorizado a realizar las siguientes tareas: PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a AWS IoT Greengrass.

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 el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en AWS IoT Greengrass. 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 administrador. AWS El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Soy administrador y quiero permitir el acceso de otras personas AWS IoT Greengrass

Para permitir el acceso de otras personas AWS IoT Greengrass, debes conceder permiso a las personas o aplicaciones que necesitan acceso. Si lo utiliza AWS IAM Identity Center para administrar personas y aplicaciones, debe asignar conjuntos de permisos a los usuarios o grupos para definir su nivel de acceso. Los conjuntos de permisos crean y asignan automáticamente políticas de IAM a las funciones de IAM asociadas a la persona o aplicación. Para obtener más información, consulte los conjuntos de permisos en la Guía del AWS IAM Identity Center usuario.

Si no utiliza el Centro de identidad de IAM, debe crear entidades de IAM (usuarios o roles) para las personas o aplicaciones a las que necesitan acceso. A continuación, debe asociar una política a la entidad que le conceda los permisos correctos en AWS IoT Greengrass. Una vez concedidos los permisos, proporcione las credenciales al usuario o al desarrollador de la aplicación. Utilizarán esas credenciales para acceder AWS. Para obtener más información sobre la creación de usuarios, grupos, políticas y permisos de IAM, consulte Identidades, políticas y permisos de IAM en IAM en la Guía del usuario de IAM.

Quiero permitir que personas ajenas a mí accedan a mis recursos Cuenta de AWSAWS IoT Greengrass

Puede crear una función de IAM que los usuarios de otras cuentas o personas ajenas a su organización puedan utilizar para acceder a sus AWS recursos. Puede especificar una persona de confianza para que asuma el rol. Para obtener más información, consulte Proporcionar acceso a un usuario de IAM en otro Cuenta de AWS usuario de su propiedad y Proporcionar acceso a cuentas de Amazon Web Services propiedad de terceros en la Guía del usuario de IAM.

AWS IoT Greengrass no admite el acceso entre cuentas en función de políticas basadas en recursos o listas de control de acceso (ACL).

Validación de conformidad en AWS IoT Greengrass

Para saber si uno Servicio de AWS está dentro del ámbito de aplicación de programas de cumplimiento específicos, consulte Servicios de AWS Alcance por programa de cumplimiento

Servicios de AWS de cumplimiento y elija el programa de cumplimiento que le interese. Para obtener información general, consulte Programas de AWS cumplimiento > Programas AWS .

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact .

Su responsabilidad de cumplimiento al Servicios de AWS utilizarlos viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. AWS proporciona los siguientes recursos para ayudar con el cumplimiento:

- Guías de inicio rápido sobre seguridad y cumplimiento: estas guías de implementación analizan las consideraciones arquitectónicas y proporcionan los pasos para implementar entornos básicos centrados en AWS la seguridad y el cumplimiento.
- Diseñando una arquitectura basada en la HIPAA seguridad y el cumplimiento en Amazon Web Services: en este documento técnico se describe cómo pueden utilizar las empresas AWS para crear HIPAA aplicaciones aptas.

  > **Note**
  >
  > No todos son aptos. Servicios de AWS HIPAA Para obtener más información, consulta la Referencia de servicios HIPAA aptos.

- AWS Recursos de de cumplimiento: esta colección de libros de trabajo y guías puede aplicarse a su industria y ubicación.
- AWS Guías de cumplimiento para clientes: comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. En las guías se resumen las mejores prácticas para garantizar la seguridad Servicios de AWS y se orientan a los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST), el Consejo de Normas de Seguridad del Sector de Tarjetas de Pago (PCI) y la Organización Internacional de Normalización (ISO)).
- Evaluación de los recursos con reglas en la guía para AWS Config desarrolladores: el AWS Config servicio evalúa en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normas.
- AWS Security Hub— Esto Servicio de AWS proporciona una visión completa del estado de su seguridad interior AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su cumplimiento con los estándares y las prácticas recomendadas del

sector de la seguridad. Para obtener una lista de los servicios y controles compatibles, consulte la Referencia de controles de Security Hub.

- Amazon GuardDuty: Servicio de AWS detecta posibles amenazas para sus cargas de trabajo Cuentas de AWS, contenedores y datos mediante la supervisión de su entorno para detectar actividades sospechosas y maliciosas. GuardDuty puede ayudarlo a cumplir con varios requisitos de conformidad, por ejemplo PCIDSS, cumpliendo con los requisitos de detección de intrusiones exigidos por ciertos marcos de cumplimiento.
- AWS Audit Manager— Esto le Servicio de AWS ayuda a auditar continuamente su AWS consumo para simplificar la gestión del riesgo y el cumplimiento de las normativas y los estándares del sector.

Resiliencia en AWS IoT Greengrass

La infraestructura global de AWS se basa en regiones y zonas de disponibilidad de Amazon Web Services. Cada región de Región de AWS proporciona 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 zonas de disponibilidad y las regiones de Amazon Web Services, consulte Infraestructura global de AWS.

Además de la infraestructura global de AWS, AWS IoT Greengrass ofrece varias características que le ayudan con sus necesidades de resiliencia y copia de seguridad de los datos.

- Si el núcleo pierde conectividad a Internet, los dispositivos cliente pueden seguir comunicándose a través de la red local.
- Puede configurar el núcleo para almacenar mensajes sin procesar orientados a destinos de Nube de AWS en una caché de almacenamiento local en lugar del almacenamiento en memoria. La caché de almacenamiento local puede persistir durante los reinicios del núcleo (por ejemplo, después de una implementación de grupo o un reinicio del dispositivo), por lo que AWS IoT Greengrass puede seguir procesando los mensajes destinados a AWS IoT Core. Para obtener más información, consulte the section called “Cola de mensajes MQTT”.
- Puede configurar el núcleo para establecer una sesión persistente con el broker de mensajes de AWS IoT Core. Esto permite que el núcleo reciba mensajes enviados mientras el núcleo está fuera

de línea. Para obtener más información, consulte the section called “Sesiones persistentes de MQTT con AWS IoT Core”.

- Puede configurar un grupo de Greengrass para escribir registros en el sistema de archivos local y en CloudWatch Logs. Si el núcleo pierde conectividad, el registro local puede continuar, pero los registros de CloudWatch se envían con un número limitado de reintentos. Cuando se agotan los reintentos, el evento se elimina. También debe tener en cuenta las limitaciones de registro.
- Puede crear funciones de Lambda que lean secuencias del administrador de secuencias y envíen los datos a destinos de almacenamiento locales.

Seguridad de la infraestructura en AWS IoT Greengrass

Como servicio gestionado, AWS IoT Greengrass 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.

Utiliza las API llamadas AWS publicadas para acceder a AWS IoT Greengrass través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Necesitamos TLS 1.2 y recomendamos TLS 1.3.
- Cifre suites con perfecto secreto (PFS), como (Ephemeral Diffie-Hellman) o DHE 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 firmarse con un identificador de clave de acceso y una clave de acceso secreta asociada a un director. IAM También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

En un AWS IoT Greengrass entorno, los dispositivos utilizan certificados X.509 y claves criptográficas para conectarse y autenticarse en él. Nube de AWS Para obtener más información, consulte the section called “Autenticación y autorización de dispositivos”.

Configuración y análisis de vulnerabilidades en AWS IoT Greengrass

Los entornos de IoT pueden constar de un gran número de dispositivos que tienen diversas capacidades, son de larga duración y están distribuidos geográficamente. Estas características hacen que la configuración del dispositivo sea compleja y propensa a errores. Y dado que los dispositivos a menudo están limitados en potencia informática, memoria y capacidades de almacenamiento, esto limita el uso del cifrado y otras formas de seguridad en los propios dispositivos. Además, los dispositivos a menudo usan software con vulnerabilidades conocidas. Estos factores hacen que los dispositivos de IoT sean un objetivo atractivo para los piratas informáticos y dificultan la protección continuada de los mismos.

Para afrontar estos desafíos, AWS IoT Device Defender proporciona herramientas para identificar los problemas de seguridad y las desviaciones de las prácticas recomendadas. Puede utilizar AWS IoT Device Defender para analizar, auditar y monitorear los dispositivos conectados para detectar comportamientos anómalos y mitigar riesgos de seguridad. AWS IoT Device Defender puede auditar dispositivos para asegurarse de que cumplen las prácticas de seguridad recomendadas y detectan comportamientos anómalos en los dispositivos. Esto permite aplicar políticas de seguridad coherentes en todos los dispositivos y responder rápidamente cuando los dispositivos sufren ataques. En las conexiones con AWS IoT Core, AWS IoT Greengrass genera identificadores de cliente predecibles que se pueden utilizar con las características de AWS IoT Device Defender. Para obtener más información, consulte AWS IoT Device Defender en la Guía para desarrolladores de AWS IoT Core.

En los entornos de AWS IoT Greengrass, debe tener en cuenta las siguientes consideraciones:

- Es su responsabilidad proteger los dispositivos físicos, el sistema de archivos en sus dispositivos y la red local.
- AWS IoT Greengrass no aplica el aislamiento de red para las funciones de Lambda definidas por el usuario, tanto si se ejecutan en un contenedor de Greengrass como si no. Por lo tanto, es posible que las funciones de Lambda se comuniquen con cualquier otro proceso que se ejecute en el sistema o fuera a través de la red.

Si pierde el control de un dispositivo del núcleo de Greengrass y desea evitar que los dispositivos cliente transmitan datos al núcleo, haga lo siguiente:

1. Quite el núcleo de Greengrass del grupo Greengrass.

2. Rote el certificado de entidad de certificación del grupo. En la consola de AWS IoT, puede rotar el certificado de entidad de certificación en la página Configuración del grupo. En la AWS IoT Greengrass API, puedes usar la CreateGroupCertificateAuthorityacción.

   También recomendamos utilizar el cifrado de disco completo si el disco duro de su dispositivo del núcleo es vulnerable al robo.

AWS IoT Greengrass y puntos de enlace de la VPC de interfaz (AWS PrivateLink)

Puede establecer una conexión privada entre la VPC y el plano de control deAWS IoT Greengrass mediante la creación de un punto de conexión de VPC de interfaz. Puede usar este punto de conexión para administrar grupos, funciones de Lambda, implementaciones y otros recursos del servicio AWS IoT Greengrass. Los puntos de conexión de interfaz cuentan con tecnología de AWS PrivateLink que le permite acceder de forma privada a las API de AWS IoT Greengrass sin una puerta de enlace de Internet, un dispositivo NAT, una conexión de VPN o una conexión de AWS Direct Connect. Las instancias de la VPC no necesitan direcciones IP públicas para comunicarse con las API de AWS IoT Greengrass. El tráfico entre la VPC y AWS IoT Greengrass no sale de la red de Amazon.

> **Note**
>
> Actualmente, no puede configurar los dispositivos principales de Greengrass para que funcionen completamente dentro de su VPC.

Cada punto de enlace de la interfaz está representado por una o más interfaces de red elásticas en las subredes.

Para obtener más información, consulte Puntos de enlace de la VPC de interfaz (AWS PrivateLink) en la Guía del usuario de Amazon VPC.

Temas

- Consideraciones para los puntos de conexión de VPC de AWS IoT Greengrass
- Crear un punto de conexión de VPC de interfaz para operaciones del plano de control AWS IoT Greengrass
- Creación de una política de puntos de enlace de la VPC para AWS IoT Greengrass

Consideraciones para los puntos de conexión de VPC de AWS IoT Greengrass

Antes de configurar un punto de conexión de VPC de interfaz para AWS IoT Greengrass, revise las Propiedades y limitaciones de los puntos de conexión de interfaz en la Guía del usuario de Amazon VPC. Además, tenga en cuenta las siguientes consideraciones:

- AWS IoT Greengrass admite realizar llamadas a todas sus acciones de la API de plano de control desde su VPC. El plano de control incluye operaciones como CreateDeployment y StartBulkDeployment. El plano de control no incluye operaciones como GetDeployment y Discover, que son operaciones del plano de datos.
- Los puntos de conexión de VPC para AWS IoT Greengrass actualmente no se admiten en las regiones de AWS de China.

Crear un punto de conexión de VPC de interfaz para operaciones del plano de control AWS IoT Greengrass

Puede crear un punto de conexión de VPC para el plano de control AWS IoT Greengrass mediante la consola de Amazon VPC o la AWS Command Line Interface (AWS CLI). Para obtener más información, consulte Creación de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Cree un punto de enlace de la VPC para AWS IoT Greengrass, mediante el siguiente nombre de servicio:

- com.amazonaws.*region*.greengrass

Si habilita DNS privado para el punto de enlace, puede realizar solicitudes a la API para AWS IoT Greengrass usando su nombre de DNS predeterminado para la región, por ejemplo `greengrass.us-east-1.amazonaws.com`. El DNS privado está habilitado de forma predeterminada.

Para obtener más información, consulte Acceso a un servicio a través de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Creación de una política de puntos de enlace de la VPC para AWS IoT Greengrass

Puede adjuntar una política de punto de conexión a su punto de conexión de VPC que controle el acceso a las operaciones del plano de control AWS IoT Greengrass. La política especifica la siguiente información:

- La entidad principal que puede realizar acciones.
- Acciones que la entidad principal puede realizar.
- Los recursos sobre los que la entidad principal puede realizar acciones.

Para obtener más información, consulte Control del acceso a los servicios con puntos de conexión de VPC en la guía del usuario de Amazon VPC.

Example Ejemplo: política de punto de enlace de la VPC para acciones de AWS IoT Greengrass

A continuación, se muestra un ejemplo de una política de puntos de enlace de AWS IoT Greengrass. Cuando se asocia con un punto de conexión, esta política concede acceso a las acciones de AWS IoT Greengrass mostradas para todas las entidades principales en todos los recursos.

```
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:StartBulkDeployment"
            ],
            "Resource": "*"
        }
    ]
}
```

Prácticas recomendadas de seguridad para AWS IoT Greengrass

Este tema contiene las prácticas de seguridad recomendadas para AWS IoT Greengrass.

Conceda los mínimos permisos posibles

Siga el principio de privilegios mínimos utilizando el conjunto mínimo de permisos en los roles de IAM. Limite el uso del comodín `*` para las propiedades `Action` y `Resource` en las políticas de IAM. En su lugar, declare un conjunto finito de acciones y recursos cuando sea posible. Para obtener más información acerca de los privilegios mínimos y otras prácticas recomendadas de política, consulte the section called “Prácticas recomendadas relativas a políticas”.

La práctica recomendada de privilegios mínimos también se aplica a las políticas de AWS IoT que asocie al núcleo de Greengrass y a los dispositivos cliente.

No codifique las credenciales de forma rígida en funciones de Lambda

No codifique las credenciales en las funciones de Lambda definidas por el usuario. Para proteger mejor sus credenciales:

- Para interactuar con los servicios de AWS, defina permisos para acciones y recursos específicos en el rol de grupo Greengrass.
- Use secretos locales para almacenar sus credenciales. O bien, si la característica utiliza el SDK de AWS, utilice las credenciales de la cadena de proveedores de credenciales predeterminada.

No registre información confidencial

Debe evitar el registro de credenciales y otra información de identificación personal (PII). Le recomendamos que implemente las siguientes medidas de seguridad, aunque el acceso a los registros locales en un dispositivo principal requiera privilegios de root y el acceso a CloudWatch los registros requiera permisos de IAM.

- No utilice información confidencial en las rutas de temas de MQTT.
- No utilice información confidencial en nombres, tipos y atributos de dispositivo (objeto) en el registro de AWS IoT Core.
- No registre información confidencial en las funciones de lambda definidas por el usuario.
- No utilice información confidencial en los nombres e identificadores de los recursos de Greengrass:
  - Connectors
  - Núcleos
  - Dispositivos

- Funciones
- Grupos
- Loggers
- Recursos (local, machine learning o secreto)
- Suscripciones

Cree suscripciones con fines específicos

Las suscripciones controlan el flujo de información en un grupo de Greengrass definiendo cómo se intercambian los mensajes entre servicios, dispositivos y funciones de Lambda. Para asegurarse de que una aplicación puede hacer solo lo que se pretende que haga, las suscripciones deben permitir a los editores enviar mensajes solo a temas específicos y limitar a los suscriptores a recibir solo mensajes de aquellos temas que sean necesarios para su funcionalidad.

Mantenga sincronizado el reloj del dispositivo

Es importante que la hora del dispositivo sea precisa. Los certificados X.509 tienen una fecha y una hora de caducidad. El reloj del dispositivo se utiliza para comprobar que un certificado de servidor sigue siendo válido. Los relojes de dispositivos pueden variar con el tiempo o las baterías pueden descargarse.

Para obtener más información, consulte las prácticas recomendadas Mantener sincronizado el reloj del dispositivo en la Guía del desarrollador de AWS IoT Core.

Administre la autenticación de dispositivos con el núcleo de Greengrass

Los dispositivos cliente pueden ejecutar FreeRTOS o utilizar el SDK de AWS IoT de dispositivos o la API de AWS IoT Greengrass de descubrimiento para obtener la información de descubrimiento utilizada para conectarse y autenticarse con el núcleo en el mismo grupo Greengrass. La información de detección incluye:

- Información de conectividad para el núcleo de Greengrass que está en el mismo grupo de Greengrass que el dispositivo cliente. Esta información incluye la dirección del host y el número de puerto de cada punto de enlace para el dispositivo del núcleo.
- Certificado de entidad de certificación del grupo utilizado para firmar el certificado de servidor MQTT local. Los dispositivos cliente utilizan el certificado de entidad de certificación del grupo para validar el certificado de servidor MQTT presentado por el núcleo.

Las siguientes son las prácticas recomendadas para que los dispositivos cliente administren la autenticación mutua con un núcleo de Greengrass. Estas prácticas pueden ayudar a mitigar el riesgo si su dispositivo del núcleo está en peligro.

Validar el certificado de servidor MQTT local para cada conexión.

Los dispositivos cliente deben validar el certificado de servidor MQTT presentado por el núcleo cada vez que establecen una conexión con el núcleo. Esta validación es el lado del dispositivo cliente de la autenticación mutua entre un dispositivo del núcleo y los dispositivos cliente. El dispositivo cliente debe poder detectar el error y terminar la conexión.

No codificar la información de detección.

Los dispositivos cliente deben depender de las operaciones de detección para obtener información de conectividad del núcleo y el certificado de entidad de certificación del grupo, incluso si el núcleo utiliza una dirección IP estática. Los dispositivos cliente no deben codificar esta información de detección.

Actualice periódicamente la información de detección.

Los dispositivos cliente deben ejecutar periódicamente la detección para actualizar la información de conectividad del núcleo y el certificado de entidad de certificación del grupo. Recomendamos que los dispositivos cliente actualicen esta información antes de establecer una conexión con el núcleo. Debido a que las duraciones más cortas entre las operaciones de detección pueden minimizar el tiempo de exposición potencial, recomendamos que los dispositivos cliente se desconecten y se vuelvan a conectar periódicamente para activar la actualización.

Si pierde el control de un dispositivo del núcleo de Greengrass y desea evitar que los dispositivos cliente transmitan datos al núcleo, haga lo siguiente:

1. Quite el núcleo de Greengrass del grupo Greengrass.
2. Rote el certificado de entidad de certificación del grupo. En la consola de AWS IoT, puede rotar el certificado de entidad de certificación en la página Configuración del grupo. En la AWS IoT Greengrass API, puedes usar la CreateGroupCertificateAuthorityacción.

   También recomendamos utilizar el cifrado de disco completo si el disco duro de su dispositivo del núcleo es vulnerable al robo.

Para obtener más información, consulte the section called “Autenticación y autorización de dispositivos”.

Véase también

- Prácticas recomendadas de seguridad para AWS IoT Core en la Guía del desarrollador de AWS IoT
- Diez reglas de oro de seguridad para soluciones de IoT industriales en el blog oficial Internet of Things on AWS

Registro y monitoreo en AWS IoT Greengrass

La monitorización es una parte importante del mantenimiento de la fiabilidad, la disponibilidad y el rendimiento de AWS IoT Greengrass y sus soluciones de AWS. Debe recopilar datos de monitoreo de todas las partes de su solución de AWS para que le resulte más sencillo depurar cualquier error que se produzca en distintas partes del código, en caso de que ocurra. Antes de empezar a monitorear AWS IoT Greengrass, debe crear un plan de monitoreo que incluya respuestas a las siguientes preguntas:

- ¿Cuáles son los objetivos de la monitorización?
- ¿Qué recursos va a monitorizar?
- ¿Con qué frecuencia va a monitorizar estos recursos?
- ¿Qué herramientas de monitorización va a utilizar?
- ¿Quién se encargará de realizar las tareas de monitoreo?
- ¿Quién debería recibir una notificación cuando surjan problemas?

Herramientas de monitoreo

AWS proporciona herramientas que puede utilizar para monitorear AWS IoT Greengrass. Puede configurar algunas de estas herramientas para que realicen la monitorización por usted. Algunas de las herramientas requieren intervención manual. Le recomendamos que automatice las tareas de monitorización en la medida de lo posible.

Puede utilizar las siguientes herramientas de monitorización automatizada para monitorizar AWS IoT Greengrass e informar de los problemas:

- Amazon CloudWatch Logs: monitoree, almacene y obtenga acceso a los archivos de registro de AWS CloudTrail u otras fuentes. Para obtener más información, consulte Monitoreo de archivos de registro en la guía del usuario de Amazon CloudWatch.
- Monitoreo de registros de AWS CloudTrail: comparta archivos de registro entre cuentas, monitoree los archivos de registro de CloudTrail en tiempo real enviándolos a CloudWatch Logs, escriba aplicaciones de procesamiento de registros en Java y compruebe que los archivos de registro no hayan cambiado después de que CloudTrail los entregara. Para obtener más información, consulte Uso de archivos de registro de CloudTrail en la Guía del usuario de AWS CloudTrail.

- Amazon EventBridge: utilice las reglas de eventos de EventBridge para recibir notificaciones sobre los cambios de estado de sus implementaciones grupales de Greengrass o las llamadas a la API registradas con CloudTrail. Para obtener más información, consulte the section called “Obtención de notificaciones de implementación” o ¿Qué es Amazon EventBridge? en la Guía del usuario de Amazon EventBridge.
- Telemetría de salud del sistema Greengrass: suscríbase para recibir los datos de telemetría enviados desde el núcleo de Greengrass. Para obtener más información, consulte the section called “Recopilación de datos de telemetría de estado del sistema”.
- Comprobación de estado local: utilice las API de estado para obtener una instantánea del estado de los procesos locales AWS IoT Greengrass en el dispositivo principal. Para obtener más información, consulte the section called “Llamar a la API de comprobación de estado local”.

Véase también

- the section called “Monitorización con registros de AWS IoT Greengrass”
- the section called “Registro de llamadas a la API de AWS IoT Greengrass con AWS CloudTrail”
- the section called “Obtención de notificaciones de implementación”

Monitorización con registros de AWS IoT Greengrass

AWS IoT Greengrass consta del servicio de nube y el software de AWS IoT Greengrass Core. El software AWS IoT Greengrass Core puede escribir registros en Amazon CloudWatch y en el sistema de archivos local del dispositivo principal. Las funciones y los conectores Lambda que se ejecutan en el núcleo también pueden escribir registros en los CloudWatch registros y en el sistema de archivos local. Puede utilizar registros para monitorizar eventos y solucionar problemas. Todas las entradas de registro de AWS IoT Greengrass incluyen una marca temporal, un nivel de registro e información sobre el evento. Los cambios en la configuración de registro surten efecto después de implementar el grupo.

El registro está configurado en el nivel de grupo. Para ver los pasos que muestran cómo configurar el registro para un grupo de Greengrass, consulte the section called “Configuración de registro en AWS IoT Greengrass”.

Acceder a CloudWatch los registros

Si configuras el CloudWatch registro, puedes ver los registros en la página Logs de la CloudWatch consola de Amazon. Los grupos de registros de AWS IoT Greengrass utilizan las siguientes convenciones de nomenclatura:

```
/aws/greengrass/GreengrassSystem/greengrass-system-component-name
/aws/greengrass/Lambda/aws-region/account-id/lambda-function-name
```

Cada grupo de registros contiene flujos de registros que utilizan la siguiente convención de nomenclatura:

```
date/account-id/greengrass-group-id/name-of-core-that-generated-log
```

Al utilizar CloudWatch Logs, se tienen en cuenta las siguientes consideraciones:

- Los registros se envían a CloudWatch los registros con un número limitado de reintentos en caso de que no haya conexión a Internet. Cuando se agotan los reintentos, el evento se elimina.
- Se aplican limitaciones de transacciones y de memoria, entre otras. Para obtener más información, consulte the section called “Limitaciones de registro”.
- Su rol de grupo de Greengrass debe permitir AWS IoT Greengrass escribir en Logs. CloudWatch Para conceder permisos, integre la siguiente política en línea en su rol de grupo.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
```

```
}
```

> **Note**
>
> Puede conceder acceso granular a los recursos de registro. Para obtener más información, consulte Uso de políticas basadas en la identidad (políticas de IAM) para CloudWatch los registros en la Guía del usuario de Amazon CloudWatch .

El rol del grupo es un rol de IAM que usted crea y asocia a su grupo de Greengrass. Puede usar la consola o la API de AWS IoT Greengrass para administrar el rol del grupo.

Uso de la consola

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Seleccione el grupo de destino.
3. Elija Ver configuración. En Rol del grupo, puede ver, asociar o desasociar el rol de grupo.

   Para ver los pasos que muestran cómo asociar el rol del grupo, consulte rol de grupo.

Uso de la CLI

- Para encontrar el rol del grupo, usa el comando. get-associated-role
- Para adjuntar el rol de grupo, utilice el associate-role-to-groupcomando.
- Para eliminar el rol de grupo, utilice el disassociate-role-from-groupcomando.

Para obtener información sobre cómo obtener el ID del grupo para utilizarlo con estos comandos, consulte the section called “Obtener el ID del grupo”.

Acceso a los registros del sistema de archivos

Si configura el registro del sistema de archivos, los archivos de registro se almacenan en *greengrass-root*/ggc/var/log en el dispositivo del núcleo. A continuación, se muestra la estructura de directorios de alto nivel:

```
greengrass-root/ggc/var/log
    - crash.log
    - system
        - log files for each Greengrass system component
    - user
        - region
            - account-id
                - log files generated by each user-defined Lambda function
            - aws
                - log files generated by each connector
```

> **Note**
>
> De forma predeterminada, *greengrass-root* es el directorio /greengrass. Si se configura un directorio de escritura, entonces los registros están en ese directorio.

Las siguientes consideraciones se aplican cuando se utilizan los registros del sistema de archivos:

- La lectura de los registros de AWS IoT Greengrass en el sistema de archivos requiere permisos raíz.
- AWS IoT Greengrass admite la rotación basada en tamaño y la limpieza automática cuando la cantidad de datos de registro se acerca al límite configurado.
- El archivo crash.log solo está disponible en los registros del sistema de archivos. Este registro no se escribe en CloudWatch Logs.
- Se aplican limitaciones de uso de disco. Para obtener más información, consulte the section called “Limitaciones de registro”.

> **Note**
>
> Los registros del software de AWS IoT Greengrass Core v1.0 se almacenan en el directorio *greengrass-root*/var/log.

Configuración de registro predeterminada

Si las opciones de registro no se configuran de forma explícita, AWS IoT Greengrass utiliza la siguiente configuración de registro predeterminada después de la implementación del primer grupo.

Componentes del sistema de AWS IoT Greengrass

- Escriba - `FileSystem`
- Componente - `GreengrassSystem`
- Nivel - `INFO`
- Espacio - `128 KB`

Funciones de Lambda definidas por el usuario

- Escriba - `FileSystem`
- Componente - `Lambda`
- Nivel - `INFO`
- Espacio - `128 KB`

> **Note**
>
> Antes de la primera implementación, únicamente los componentes del sistema escriben registros en el sistema de archivos, ya que no se han implementado funciones de Lambda definidas por el usuario.

Configuración de registro en AWS IoT Greengrass

Puede utilizar la consola AWS IoT o las API de AWS IoT Greengrass para configurar el registro AWS IoT Greengrass.

> **Note**
>
> AWS IoT GreengrassPara permitir la escritura de registros en los CloudWatch registros, su rol de grupo debe permitir las acciones de CloudWatch registro requeridas.

Configuración de registro (consola)

Puede configurar el registro en la página Configuración del grupo.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija el grupo en el que desea configurar el registro.
3. En la página de configuración del grupo, elija la pestaña Registros.
4. Elija la ubicación de registro de la siguiente manera:

   - Para configurar el CloudWatch registro, para configurar CloudWatch los registros, seleccione Editar.
   - Para configurar el registro en el sistema de archivos, en Local logs configuration (Configuración de registros locales), elija Edit (Editar).

   Puede configurar el registro para una ubicación o para ambas.
5. En el modal de configuración de edición de registros, seleccione el nivel de registro del sistema Greengrass o el nivel de registro de las funciones de Lambda de usuario. Puede elegir un componente o ambos.
6. Elija el nivel de eventos más bajo que desea registrar. Los eventos por debajo de este umbral se filtran y no se almacenan.
7. Seleccione Guardar. Los cambios surten efecto después de implementar el grupo.

Configuración de registro (API)

Puede utilizar API de registrador de AWS IoT Greengrass para configurar el registro mediante programación. Por ejemplo, utilice la acción `CreateLoggerDefinition` para crear una definición de registradores basada en una carga `LoggerDefinitionVersion`, que utiliza la sintaxis siguiente:

```
{
  "Loggers": [
    {
      "Id": "string",
      "Type": "FileSystem|AWSCloudWatch",
      "Component": "GreengrassSystem|Lambda",
      "Level": "DEBUG|INFO|WARN|ERROR|FATAL",
      "Space": "integer"
    },
    {
      "Id": "string",
      ...
    }
  ]
}
```

`LoggerDefinitionVersion` es una matriz de uno o varios objetos `Logger` que tienen las propiedades siguientes:

`Id`

Un identificador para el registrador.

`Type`

El mecanismo de almacenamiento de eventos de registro. Cuando `AWSCloudWatch` se usa, los eventos del registro se envían a CloudWatch los registros. Cuando se utiliza `FileSystem`, los eventos de registro se almacenan en el sistema de archivos local.

Valores válidos: `AWSCloudWatch`, `FileSystem`

`Component`

El origen del evento de registro. Cuando se utiliza `GreengrassSystem`, se registran eventos de componentes del sistema de Greengrass. Cuando se utiliza `Lambda`, se registran los eventos de las funciones de Lambda definidas por el usuario.

Valores válidos: `GreengrassSystem`, `Lambda`

`Level`

El umbral del nivel de registro. Los eventos de registro por debajo de este umbral se filtran y no se almacenan.

Valores válidos: `DEBUG`, `INFO` (recomendado), `WARN`, `ERROR`, `FATAL`

`Space`

La cantidad máxima de almacenamiento local, en KB, que se utilizará para almacenar registros. Este campo se aplica únicamente cuando `Type` se establece en `FileSystem`.

Ejemplo de configuración

En el siguiente ejemplo de `LoggerDefinitionVersion` se especifica una configuración de registro que:

- Activa el registro del nivel `ERROR` en el sistema de archivos para los componentes del sistema de AWS IoT Greengrass.
- Activa el registro del nivel `INFO` (y superior) en el sistema de archivos para las funciones de Lambda definidas por el usuario.
- Activa CloudWatch `INFO` (y más) el registro para las funciones Lambda definidas por el usuario.

```
{
  "Name": "LoggingExample",
  "InitialVersion": {
    "Loggers": [
      {
        "Id": "1",
        "Component": "GreengrassSystem",
        "Level": "ERROR",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "2",
        "Component": "Lambda",
        "Level": "INFO",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "3",
        "Component": "Lambda",
        "Level": "INFO",
        "Type": "AWSCloudWatch"
```

```
      }
    ]
  }
}
```

Una vez que se haya creado una versión de la definición de registradores, puede utilizar su ARN de la versión para crear una versión del grupo antes de implementar este.

Limitaciones de registro

AWS IoT Greengrass tiene las siguientes limitaciones de registro.

Transacciones por segundo

Cuando el registro CloudWatch está activado, el componente de registro registra los eventos de forma local por lotes antes de enviarlos a CloudWatch, de modo que puede registrar a una velocidad superior a cinco solicitudes por segundo por flujo de registro.

Memoria

Si AWS IoT Greengrass está configurada para enviar registros CloudWatch y una función Lambda registra más de 5 MB/segundo durante un período prolongado, la canalización de procesamiento interno finalmente se llena. El peor caso teórico es 6 MB por función de Lambda.

Desfase del reloj

Cuando el inicio de sesión CloudWatch está activado, el componente de registro firma las solicitudes CloudWatch mediante el proceso de firma normal de la versión 4 de Signature. Si la hora del sistema del dispositivo principal de AWS IoT Greengrass no está sincronizada en más de 15 minutos, las solicitudes se rechazan.

Consumo de disco

Utilice la siguiente fórmula para calcular la cantidad máxima total de consumo de disco para la actividad de registro.

```
greengrass-system-component-space * 8    // 7 if automatic IP detection is disabled
  + 128KB                                // the internal log for the local logging
 component
```

```
  + lambda-space * lambda-count          // different versions of a Lambda function are
treated as one
```

Donde:

`greengrass-system-component-space`

La cantidad máxima de almacenamiento local para los registros de los componentes del sistema de AWS IoT Greengrass.

`lambda-space`

La cantidad máxima de almacenamiento local para registros de funciones de Lambda.

`lambda-count`

El número de funciones de Lambda implementadas.

Pérdida de registros

Si tu dispositivo AWS IoT Greengrass principal está configurado para iniciar sesión únicamente CloudWatch y no hay conexión a Internet, no tienes forma de recuperar los registros que se encuentran actualmente en la memoria.

Cuando se finalizan las funciones de Lambda (por ejemplo, durante el despliegue), no se escriben los registros de unos segundos. CloudWatch

CloudTrail registros

AWS IoT Greengrass se ejecuta con AWS CloudTrail, un servicio que proporciona un registro de las acciones hechas por un usuario, un rol o un servicio de AWS en AWS IoT Greengrass. Para obtener más información, consulte the section called “Registro de llamadas a la API de AWS IoT Greengrass con AWS CloudTrail”.

Registro de llamadas a la API de AWS IoT Greengrass con AWS CloudTrail

AWS IoT Greengrassestá integrado con AWS CloudTrail un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un AWS servicio enAWS IoT Greengrass. CloudTrail captura todas las llamadas a la API AWS IoT Greengrass como eventos. Las llamadas capturadas

incluyen las llamadas desde la consola de AWS IoT Greengrass y las llamadas desde el código a las operaciones de la API de AWS IoT Greengrass. Si crea una ruta, puede habilitar la entrega continua de CloudTrail eventos a un bucket de Amazon S3, incluidos los eventos paraAWS IoT Greengrass. Si no configura una ruta, podrá ver los eventos más recientes en la CloudTrail consola, en el historial de eventos. Con la información recopilada por usted CloudTrail, puede determinar a AWS IoT Greengrass qué dirección IP se realizó la solicitud, quién la realizó, cuándo se realizó y detalles adicionales.

Para obtener más información CloudTrail, consulte la Guía AWS CloudTrail del usuario.

AWS IoT Greengrassinformación en CloudTrail

CloudTrail está habilitada en tu cuenta Cuenta de AWS al crear la cuenta. Cuando se produce una actividad enAWS IoT Greengrass, esa actividad se registra en un CloudTrail evento junto con otros eventos de AWS servicio en el historial de eventos. Puede ver, buscar y descargar eventos recientes en su Cuenta de AWS. Para obtener más información, consulte Visualización de eventos con el historial de CloudTrail eventos.

Para mantener un registro continuo de eventos en la Cuenta de AWS, incluidos los eventos de AWS IoT Greengrass, cree un registro de seguimiento. Un rastro permite CloudTrail entregar 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 Región 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. Además, puede configurar otros AWS servicios para analizar más a fondo los datos de eventos recopilados en los CloudTrail registros y actuar en función de ellos. Para más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- CloudTrail servicios e integraciones compatibles
- Configuración de las notificaciones de Amazon SNS para CloudTrail
- Recibir archivos de CloudTrail registro de varias regiones y recibir archivos de CloudTrail registro de varias cuentas

Todas AWS IoT Greengrass las acciones se registran CloudTrail y se documentan en la referencia de la AWS IoT Greengrass API. Por ejemplo, las llamadas a las `CreateFunctionDefinition` acciones `AssociateServiceRoleToAccount` `GetGroupVersionGetConnectivityInfo`, y generan entradas en los archivos de CloudTrail registro.

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 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 .

Descripción de las entradas de los archivos de registro de AWS IoT Greengrass

Un rastro es una configuración que permite la entrega de eventos como archivos de registro a un bucket de Amazon S3 que usted especifique. CloudTrail Los archivos de registro contienen una o más entradas de registro. Un evento representa una solicitud única de cualquier fuente e incluye información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc. CloudTrail Los archivos de registro no son un registro ordenado de las llamadas a la API pública, por lo que no aparecen en ningún orden específico.

En el siguiente ejemplo, se muestra una entrada de CloudTrail registro que demuestra la `AssociateServiceRoleToAccount` acción.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T17:04:02Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "AssociateServiceRoleToAccount",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
```

```
    "userAgent": "apimanager.amazonaws.com",
    "errorCode": "BadRequestException",
    "requestParameters": null,
    "responseElements": {
        "Message": "That role ARN is invalid."
    },
    "requestID": "a5990ec6-d22e-11e8-8ae5-c7d2eEXAMPLE",
    "eventID": "b9070ce2-0238-451a-a9db-2dbf1EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la `GetGroupVersion` acción.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-10-17T18:14:57Z"
            }
        },
        "invokedBy": "apimanager.amazonaws.com"
    },
    "eventTime": "2018-10-17T18:15:11Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "GetGroupVersion",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "GroupVersionId": "6c477753-dbf2-4cb8-acc3-5ba4eEXAMPLE",
        "GroupId": "90fcf6df-413c-4515-93a8-00056EXAMPLE"
    },
```

```
    "responseElements": null,
    "requestID": "95dcffce-d238-11e8-9240-a3993EXAMPLE",
    "eventID": "8a608034-82ed-431b-b5e0-87fbdEXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la `GetConnectivityInfo` acción.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T17:02:12Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "GetConnectivityInfo",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "ThingName": "us-east-1_CIS_1539795000000_"
    },
    "responseElements": null,
    "requestID": "63e3ebe3-d22e-11e8-9ddd-5baf3EXAMPLE",
    "eventID": "db2260d1-a8cc-4a65-b92a-13f65EXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la `CreateFunctionDefinition` acción.

```
{
```

```
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T18:01:11Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateFunctionDefinition",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "InitialVersion": "***"
    },
    "responseElements": {
        "CreationTimestamp": "2018-10-17T18:01:11.449Z",
        "LatestVersion": "dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
        "LatestVersionArn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE/versions/dae06a61-c32c-41e9-
b983-ee5cfEXAMPLE",
        "LastUpdatedTimestamp": "2018-10-17T18:01:11.449Z",
        "Id": "7a94847d-d4d2-406c-9796-a3529EXAMPLE",
        "Arn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/
functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE"
    },
    "requestID": "a17d4b96-d236-11e8-a74e-3db27EXAMPLE",
    "eventID": "bdbf6677-a47a-4c78-b227-c5f64EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


Véase también

- ¿Qué es AWS CloudTrail? en la Guía del usuario de AWS CloudTrail
- Cómo crear una EventBridge regla que se active en una llamada a la AWS API utilizando CloudTrail la Guía del EventBridge usuario de Amazon
- Referencia de la API del AWS IoT Greengrass

Recopilación de datos de telemetría del estado del sistema desde los dispositivos AWS IoT Greengrass principales

Los datos de telemetría del estado del sistema son datos de diagnóstico que pueden ayudarlo a monitorear el rendimiento de las operaciones críticas en sus dispositivos principales de Greengrass. El agente de telemetría del núcleo de Greengrass recopila datos de telemetría locales y los publica en Amazon EventBridge sin requerir ninguna interacción con el cliente. Los dispositivos principales publican datos de telemetría en EventBridge en función del «mejor esfuerzo». Por ejemplo, es posible que los dispositivos principales no entreguen los datos de telemetría cuando están fuera de línea.

> **Note**
>
> Amazon EventBridge: es un servicio de bus de eventos que se puede utilizar para conectar las aplicaciones con datos procedentes de varias fuentes, como los dispositivos centrales de Greengrass y las notificaciones de implementación. Para obtener más información, consulte What is Amazon EventBridge? (¿Qué es Amazon EventBridge?) en la Guía del usuario de Amazon EventBridge.

Puede crear proyectos y aplicaciones para recuperar, analizar, transformar y generar informes sobre los datos de telemetría de sus dispositivos periféricos. Los expertos de dominio, como los ingenieros de procesos, pueden utilizar estas aplicaciones para obtener información sobre el estado de la flota.

Para garantizar que los componentes periféricos de Greengrass funcionen correctamente, AWS IoT Greengrass utiliza los datos con fines de desarrollo y mejora de la calidad. Esta característica también ayuda a informar sobre las capacidades periféricas nuevas y mejoradas. AWS IoT Greengrass solo conserva datos de telemetría durante un máximo de siete días.

Esta característica está disponible en la versión 1.11.0 del software AWS IoT Greengrass Core y está habilitada de forma predeterminada para todos los dispositivos principales de Greengrass, incluidos los principales existentes. Empezará a recibir datos automáticamente en cuanto se actualice a la versión 1.11.0 o posterior del software AWS IoT Greengrass Core.

Para obtener información sobre cómo acceder o administrar datos de telemetría publicados, consulte the section called “Suscribirse para recibir datos de telemetría”.

El agente de telemetría recopila y publica las siguientes métricas del sistema.

Métricas de telemetría

Nombre	Descripción	Origen
`SystemMemUsage`	La cantidad de memoria que utilizan actualmente todas las aplicaciones del dispositivo principal de Greengrass, incluido el sistema operativo.	System (Sistema)
`CpuUsage`	La cantidad de CPU que utilizan actualmente todas las aplicaciones del dispositivo principal de Greengrass, incluido el sistema operativo.	System (Sistema)
`TotalNumberOfFDs`	El número de descriptores de archivos almacenados por el sistema operativo del dispositivo principal de Greengrass. Un descriptor de archivo identifica exclusivamente un archivo abierto.	System (Sistema)
`LambdaOutOfMemory`	El número de ejecuciones que provocan que la función de Lambda se quede sin memoria.	System (Sistema)
`DroppedMessageCount`	El número de mensajes descartados que están destinados a AWS IoT Core.	Componente del sistema de `GGCloudSpooler`
`LambdaTimeout`	Número de tiempos de espera para ejecutar la función Lambda definida por el usuario.	Función de Lambda definido por el usuario, Nube de AWS y sistema

Nombre	Descripción	Origen
`LambdaUngracefullyKilled`	El número de ejecuciones que la función de Lambda definida por el usuario no completa.	Función de Lambda definido por el usuario, Nube de AWS y sistema
`LambdaError`	El número de ejecuciones que dan como resultado los registros de errores de escritura de la función de Lambda definida por el usuario.	Función de Lambda definido por el usuario, Nube de AWS y sistema
`BytesAppended`	El número de bytes de datos anexos al administrador de secuencias.	Componente del sistema de `GGStreamManager`
`BytesUploadedToIoTAnalytics`	El número de bytes de datos que el administrador de secuencias exporta a los canales en AWS IoT Analytics.	Componente del sistema de `GGStreamManager`
`BytesUploadedToKinesis`	El número de bytes de datos que el administrador de secuencias exporta a las transmisiones de Amazon Kinesis Data Streams.	Componente del sistema de `GGStreamManager`
`BytesUploadedToIoTSiteWise`	El número de bytes de datos que el administrador de secuencias exporta a las propiedades de los activos en AWS IoT SiteWise.	Componente del sistema de `GGStreamManager`
`BytesUploadedToS3ExportTaskExecutor`	El número de bytes de datos que el administrador de secuencias exporta a objetos de Amazon S3.	Componente del sistema de `GGStreamManager`

Nombre	Descripción	Origen
`BytesUploadedToHTTP`	El número de bytes de datos que el administrador de secuencias exporta a HTTP.	Componente del sistema de `GGStreamManager`

Configuración de los ajustes de telemetría

La telemetría de Greengrass utiliza los siguientes ajustes:

- El agente de telemetría agrega los datos de telemetría cada hora.
- El agente de telemetría publica un mensaje de telemetría cada 24 horas.

> **Note**
>
> Los ajustes no se pueden cambiar.

Puede habilitar o deshabilitar la característica de telemetría para un dispositivo principal de Greengrass. AWS IoT Greengrass usa sombras para administrar la configuración de telemetría. Los cambios se aplican inmediatamente cuando el dispositivo principal tiene una conexión a AWS IoT Core.

El agente de telemetría publica los datos mediante el protocolo MQTT con un nivel de calidad de servicio (QoS) de 0. Esto significa que no confirma la entrega ni reintenta publicar los intentos. Los mensajes de telemetría comparten una conexión MQTT con otros mensajes para las suscripciones destinadas para AWS IoT Core.

Además de los costes de enlace de datos, la transferencia de datos desde el núcleo hasta AWS IoT Core es gratuita. Esto se debe a que el agente publica en un tema AWS reservado. Sin embargo, en función de su caso de uso, es posible que incurra en costes cuando reciba o procese los datos.

Requisitos

Se aplican los siguientes requisitos al configurar los ajustes de telemetría:

- Debe usar software AWS IoT Greengrass Core versión 1.11.0 o posterior.

> **Note**
>
> Si está ejecutando una versión anterior y no desea usar telemetría, no tiene que realizar ninguna acción.

- Debe proporcionar permisos de IAM para actualizar la sombra principal (cosa) y para llamar a las API de configuración antes de actualizar los ajustes de telemetría.

  El siguiente ejemplo de política de IAM le permite administrar la configuración oculta y el tiempo de ejecución de un núcleo específico:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowManageShadow",
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
                "iot:DescribeThing"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        },
        {
            "Sid": "AllowManageRuntimeConfig",
            "Effect": "Allow",
            "Action": [
                "greengrass:GetCoreRuntimeConfiguration",
                "greengrass:UpdateCoreRuntimeConfiguration"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name"
            ]
        }
    ]
}
```

Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Configurar los ajustes de telemetría (consola)

A continuación se muestra cómo actualizar los ajustes de telemetría de un núcleo de Greengrass en la consola AWS IoT Greengrass.

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. En Grupos de Greengrass, elija su grupo objetivo.
3. En la página de configuración del grupo, en la sección Descripción general, elija su Greengrass core.
4. En la página de configuración del núcleo, elija la pestaña Telemetría.
5. En la sección Telemetría del estado del sistema, seleccione Configurar.
6. En Configurar telemetría, seleccione Telemetría para activar o desactivar el estado de Telemetría.

> **Important**
>
> De forma predeterminada, la característica de telemetría está habilitada para la versión 1.11.0 o posterior del software AWS IoT Greengrass Core.

Los cambios surten efecto durante el tiempo de ejecución. No es necesario implementar el grupo.

Configurar los ajustes de telemetría (CLI)

En la AWS IoT Greengrass API, el objeto `TelemetryConfiguration` representa la configuración de telemetría de un Greengrass core. Este objeto forma parte del objeto `RuntimeConfiguration` asociado al núcleo. Puede usar la AWS IoT Greengrass API, AWS CLI o el AWS SDK para administrar la telemetría de Greengrass. En los ejemplos de esta sección se utiliza la AWS CLI.

Para comprobar los ajustes de telemetría

El siguiente comando obtiene los ajustes de telemetría de un Greengrass core.

- Sustituya *core-thing-name* por el nombre del núcleo objetivo.

  Para obtener el nombre de la cosa, utilice el comando get-core-definition-version. El comando devuelve el ARN de la cosa que contiene el nombre de la cosa.

```
aws greengrass get-thing-runtime-configuration --thing-name core-thing-name
```

El comando `GetCoreRuntimeConfigurationResponse` devuelve los datos en un objeto JSON. Por ejemplo:

```
{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "OutOfSync",
            "Telemetry": "On"
        }
    }
}
```

Para configurar los ajustes de telemetría

El siguiente comando actualiza los ajustes de telemetría de un Greengrass core.

- Sustituya *core-thing-name* por el nombre del núcleo objetivo.

  Para obtener el nombre de la cosa, utilice el comando get-core-definition-version. El comando devuelve el ARN de la cosa que contiene el nombre de la cosa.

JSON expanded

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration  '{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "InSync",
            "Telemetry": "Off"
        }
    }
}
```

JSON single-line

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --telemetry-configuration "{\"TelemetryConfiguration\":{\"ConfigurationSyncStatus\":\"InSync\",\"Telemetry\":\"Off\"}}"
```

Se han aplicado cambios en la configuración a la telemetría si el `ConfigurationSyncStatus` es `InSync`. Los cambios surten efecto durante el tiempo de ejecución. No es necesario implementar el grupo.

Objeto TelemetríaConfiguración

El objeto `TelemetryConfiguration` incluye las siguientes propiedades:

`ConfigurationSyncStatus`

Comprueba si los ajustes de telemetría están sincronizados. Es posible que no realice cambios en esta propiedad.

Tipo: cadena

Valores válidos: `InSync` o `OutOfSync`

`Telemetry`

Activa o desactiva la telemetría. El valor predeterminado es `On`.

Tipo: String

Valores válidos: `On` o `Off`

Suscribirse para recibir datos de telemetría

Puede crear reglas en Amazon EventBridge que definan cómo procesar los datos de telemetría publicados desde el dispositivo Greengrass core. Cuando EventBridge recibe los datos, invoca las acciones de destino definidas en sus reglas. Por ejemplo, puede crear reglas de eventos que envíen notificaciones, almacenen información sobre eventos, adopten medidas correctivas o invoquen otros eventos.

Evento de telemetría

El evento de un cambio de estado de la implementación, incluidos los datos de telemetría, tiene el siguiente formato:

```
{
    "version": "0",
    "id": "f70f943b-9ae2-e7a5-fec4-4c22178a3e6a",
    "detail-type": "Greengrass Telemetry Data",
    "source": "aws.greengrass",
    "account": "123456789012",
    "time": "2020-07-28T20:45:53Z",
    "region": "us-west-1",
    "resources": [],
    "detail": {
        "ThingName": "CoolThing",
        "Schema": "2020-06-30",
        "ADP": [
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToKinesis",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToS3ExportTaskExecutor",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "StreamManager",
```

```
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToHTTP",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToIoTAnalytics",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToIoTSiteWise",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
                "M": [
                    {
                        "N": "LambdaTimeout",
                        "Sum": 15,
                        "U": "Count"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "CloudSpooler",
```

```
                "M": [
                    {
                        "N": "DroppedMessageCount",
                        "Sum": 15,
                        "U": "Count"
                    }
                ]
            },
            {
                "TS": 1593727692,
                "NS": "SystemMetrics",
                "M": [
                    {
                        "N": "SystemMemUsage",
                        "Sum": 11.23,
                        "U": "Megabytes"
                    },
                    {
                        "N": "CpuUsage",
                        "Sum": 35.63,
                        "U": "Percent"
                    },
                    {
                        "N": "TotalNumberOfFDs",
                        "Sum": 416,
                        "U": "Count"
                    }
                ]
            },
            {
                "TS": 1593727692,
                "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
                "M": [
                    {
                        "N": "LambdaOutOfMemory",
                        "Sum": 12,
                        "U": "Count"
                    },
                    {
                        "N": "LambdaUngracefullyKilled",
                        "Sum": 100,
                        "U": "Count"
                    },
                    {
```

```
                        "N": "LambdaError",
                        "Sum": 7,
                        "U": "Count"
                    }
                ]
            }
        ]
    }
}
```

La matriz ADP contiene una lista de puntos de datos agregados que tienen las siguientes propiedades:

`TS`

Obligatorio. Marca temporal del momento en que se agregaron los datos.

`NS`

Obligatorio. El espacio de nombre del sistema de archivos.

`M`

Obligatorio. Lista de métricas Una métrica contiene las siguientes propiedades:

`N`

El nombre de la métrica.

`Sum`

El valor métrico agregado. El agente de telemetría añade nuevos valores al total anterior, por lo que la suma es un valor cada vez mayor. Puede usar la marca de tiempo para encontrar el valor de una agregación específica. Por ejemplo, para encontrar el último valor agregado, reste el valor de la marca de tiempo anterior del último valor de marca de tiempo.

`U`

La unidad del valor métrico.

`ThingName`

Obligatorio. El nombre del dispositivo al que apunta.

Prerequisitos para crear las reglas de EventBridge

Antes de crear reglas de EventBridge de eventos para AWS IoT Greengrass, debe hacer lo siguiente:

- Familiarizase con los eventos, las reglas y los destinos de EventBridge.
- Cree y configure los destinos que las reglas de EventBridge han invocado. Las reglas pueden invocar muchos tipos de objetivos, como transmisiones de Amazon Kinesis, funciones AWS Lambda, temas de Amazon SNS y colas de Amazon SQS.

  Su regla de EventBridge y los objetivos asociados deben estar en el Región de AWS donde creó sus recursos de Greengrass. Para obtener más información, consulte Puntos de enlace y cuotas en la Referencia general de AWS.

Para obtener más información, consulte What is Amazon EventBridge? (¿Qué es Amazon EventBridge?) y Getting started with Amazon EventBridge (Introducción a Amazon EventBridge) en la Guía del usuario de Amazon EventBridge.

Cree una regla de eventos para obtener datos de telemetría (consola)

Siga los siguientes pasos para usar el AWS Management Console para crear una regla de EventBridge que reciba los datos de telemetría publicados por Greengrass core. De este modo, los servidores web, las direcciones de correo electrónico y otros suscriptores del tema podrán responder al evento. Para obtener más información, consulte Creación de una regla de EventBridge que se activa en función de un evento de un recurso de AWS en la Guía del usuario de Amazon EventBridge.

1. Abra la consola de Amazon EventBridge y elija Crear regla.
2. En Name and description (Nombre y descripción), escriba un nombre y descripción para la regla.
3. Elija bus de eventos y active la regla en el bus de eventos seleccionado.
4. Seleccione Tipo de regla y luego Regla con un patrón de evento.
5. Elija Next (Siguiente).
6. En Event source (Origen del evento), elija AWS events or EventBridge partner events (Eventos o eventos de socios de EventBridge).
7. En Evento de muestra, elija eventos AWS y seleccione Datos de telemetría de Greengrass.
8. En Patrón de eventos, seleccione las siguientes opciones:

   a. En Event source (Origen del evento), elija AWS services (Servicios de).

b. En AWSServicio, elija Greengrass.

c. En Tipo de evento, elija Datos de telemetría de Greengrass.

9. Elija Next (Siguiente).
10. En Target 1 elija AWS service.
11. En Seleccione un objetivo, elija la cola de SQS.
12. En Cola, elija su función.

Cree una regla de eventos para obtener datos de telemetría (CLI)

Siga los siguientes pasos para usar el AWS CLI para crear una regla de EventBridge que reciba los datos de telemetría publicados por Greengrass core. De este modo, los servidores web, las direcciones de correo electrónico y otros suscriptores del tema podrán responder al evento.

1. Cree la regla de .

   - Sustituya *thing-name* por el nombre de la cosa del núcleo.

     Para obtener el nombre de la cosa, utilice el comando get-core-definition-version. El comando devuelve el ARN de la cosa que contiene el nombre de la cosa.

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\":
    [\"thing-name\"]}}"
   ```

   Las propiedades que se omiten en el patrón no se tienen en cuenta.

2. Añada el tema como destino de la regla. El siguiente ejemplo usa Amazon SQS, pero puede configurar otros tipos de objetivos.

   - Sustituya *queue-arn* por el ARN de su cola de Amazon SQS.

   ```
   aws events put-targets \
     --rule TestRule \
     --targets "Id"="1","Arn"="queue-arn"
   ```

> **Note**
>
> Para permitir que Amazon EventBridge invoque la cola de destino, debe agregar en el tema una política basada en recursos. Para obtener más información, consulte Amazon SQS permissions en la Guía del usuario de Amazon EventBridge.

Para obtener más información, consulte Events and event patterns in EventBridge (Eventos y patrones de eventos en EventBridge) en la Guía del usuario de Amazon EventBridge.

Solución de problemas de telemetría AWS IoT Greengrass

Utilice la siguiente información como ayuda para solucionar problemas al configurar la telemetría AWS IoT Greengrass.

Error: la respuesta contiene «ConfigurationStatus»: «OutOfSync» después de ejecutar el comando get-thing-runtime-configuration

Soluciones:

- El servicio sombra de dispositivo AWS IoT tarda en procesar las actualizaciones de configuración en tiempo de ejecución y en entregar las actualizaciones al dispositivo principal de Greengrass. Puede esperar y comprobar si los ajustes de telemetría están sincronizados más adelante.
- Asegúrese de que el dispositivo del núcleo esté en línea.
- Active Registros de Amazon CloudWatch AWS IoT Core para monitorear la sombra.
- Utilice las métricas de AWS IoT para monitorizar su objeto .

Llamar a la API de comprobación de estado local

AWS IoT Greengrass contiene una API HTTP local que proporciona una instantánea del estado actual de los procesos de trabajo locales que fueron iniciados por AWS IoT Greengrass. Esta instantánea incluye funciones de Lambda definidas por el usuario y funciones de Lambda del sistema. Las funciones de Lambda del sistema son componentes del software de AWS IoT Greengrass Core. Se ejecutan como procesos de trabajo locales en el dispositivo principal y gestionan operaciones como el enrutamiento de mensajes, la sincronización virtual local y la detección automática de direcciones IP.

La API de comprobación de estado admite las siguientes solicitudes:

- Envíe una solicitud `GET` para obtener información sobre la salud de todos los trabajadores.
- Envíe una solicitud `POST` para obtener información sobre la salud de trabajadores específicos.

Las solicitudes se envían de forma local en el dispositivo y no requieren conexión a Internet.

Obtener información sobre la salud de todos los trabajadores

Envíe una solicitud de `GET` para obtener información sobre la salud de todos los trabajadores.

- Sustituya el *`puerto`* por el número de puerto del IPC.

```
GET http://localhost:port/2016-11-01/health/workers
```

`port`

El número de puerto del proxy.

El valor puede variar entre 1024 y 65535. El valor predeterminado es 8000.

Para cambiar este número de puerto, puede actualizar la propiedad `ggDaemonPort` del archivo `config.json`. Para obtener más información, consulte Archivo de configuración de AWS IoT Greengrass Core.

Ejemplo de solicitud

La siguiente solicitud `curl` de ejemplo obtiene información de salud de todos los trabajadores.

```
curl http://localhost:8000/2016-11-01/health/workers
```

Respuesta de JSON

Esta solicitud devuelve un conjunto de objetos de información sobre la salud de los trabajadores.

Respuesta de ejemplo

En el siguiente ejemplo de respuesta se enumeran los objetos de información de salud de todos los procesos de trabajo iniciados por AWS IoT Greengrass.

```
[
    {
      "FuncArn": "arn:aws:lambda:::function:GGShadowService:1",
      "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
      "ProcessId": "1234",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:::function:GGSecretManager:1",
      "WorkerId": "a9916cc2-1b4d-4f0e-4b12-b1872EXAMPLE",
      "ProcessId": "9798",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3",
      "WorkerId": "2e6f785e-66a5-42c9-67df-42073EXAMPLE",
      "ProcessId": "11837",
      "WorkerState": "Waiting"
    },
    ...
]
```

Obtener información sobre la salud de trabajadores específicos

Envíe una solicitud de POST para obtener información sobre la salud de trabajadores específicos. Sustituya el *puerto* por el número de puerto del IPC. El valor predeterminado es 8000.

```
POST http://localhost:port/2016-11-01/health/workers
```

Ejemplo de solicitud

La siguiente solicitud `curl` de ejemplo obtiene información de salud de trabajadores específicos.

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

A continuación se muestra un ejemplo del cuerpo de la solicitud de `body.json`:

```
{
    "FuncArns": [
        "arn:aws:lambda:::function:GGShadowService:1",
        "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
    ]
```

```
}
```

El cuerpo de la solicitud contiene una `FuncArns` matriz.

`FuncArns`

Una lista de nombres de recursos de Amazon (ARN) para las funciones de Lambda que representan a los trabajadores de destino.

- Para las funciones de Lambda definidas por el usuario, especifique el ARN de la versión actualmente implementada. Si ha agregado funciones de Lambda al grupo mediante un ARN de alias, puede usar la solicitud GET para obtener todos los trabajadores y, a continuación, elegir los ARN que desea consultar.
- Para las funciones de Lambda del sistema, especifique el ARN de la función de Lambda correspondiente. Para obtener más información, consulte the section called “Funciones de Lambda del sistema”.

Tipo: matriz de cadenas

Longitud mínima: 1

Longitud máxima: el número total de trabajadores iniciados por AWS IoT Greengrass en el dispositivo principal.

Respuesta de JSON

Esta solicitud devuelve una matriz `Workers` y una matriz `InvalidArns`.

`Workers`

Una lista de objetos de información de salud para los trabajadores especificados.

Tipo: conjunto de objetos de información de salud

`InvalidArns`

Una lista de los ARN de funciones que no son válidos, incluidos los ARN de funciones que no tienen trabajadores asociados.

Tipo: matriz de cadenas

Respuesta de ejemplo

La siguiente respuesta de ejemplo enumera los objetos de información sanitaria de los trabajadores especificados.

```
{
    "Workers": [
        {
            "FuncArn": "arn:aws:lambda:::function:GGShadowService:1",
            "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
            "ProcessId": "1234",
            "WorkerState": "Waiting"
        },
        {
            "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-
function:3",
            "WorkerId": "2e6f785e-66a5-42c9-67df-42073ESAMPLE",
            "ProcessId": "11837",
            "WorkerState": "Waiting"
        }
    ],
    "InvalidArns" : [
        "some-malformed-arn",
        "arn:aws:lambda:us-west-2:123456789012:function:some-unknown-function:1"
    ]
}
```

Esta solicitud devuelve los siguientes errores:

400 Solicitud no válida

El cuerpo de la solicitud está malformado. Para resolver este problema, utilice el siguiente formato y vuelva a enviar la solicitud:

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

400 La solicitud supera el número máximo de trabajadores

La cantidad de ARN especificada en la matriz `FuncArns` supera la cantidad de trabajadores.

Información de salud de los trabajadores

Un objeto de información sanitaria contiene las siguientes propiedades:

`FuncArn`

El ARN de la funcióna de Lambda del sistema que representa al trabajador.

Escriba: `string`

`WorkerId`

El ID del trabajador. Esto es útil para depurar prototipos. El archivo `runtime.log` y los registros de la función de Lambda contienen el ID del trabajador, por lo que esta propiedad puede resultar especialmente útil para depurar una función de Lambda bajo demanda que activa varias instancias.

Escriba: `string`

`ProcessId`

El ID de proceso (PID) del proceso de trabajo.

Escriba: `int`

`WorkerState`

El estado del trabajador.

Escriba: `string`

A continuación se indican los posibles estados de los trabajadores:

`Working`

Procesando un mensaje.

`Waiting`

Esperando un mensaje. Se aplica a las funciones de Lambda de larga duración que se ejecutan como un daemon o un proceso independiente.

`Starting`

Se puso en marcha, para empezar.

`FailedInitialization`

No se pudo inicializar.

`Terminated`

Detenido por el daemon de Greengrass

`NotStarted`

No se pudo iniciar y se realizó otro intento de inicio.

`Initialized`

Se inicializó correctamente.

Funciones de Lambda del sistema

Puede solicitar información de estado para las siguientes funciones de Lambda del sistema:

`GGCloudSpooler`

Administra la cola de los mensajes MQTT que tienen AWS IoT Core como origen o destino.

ARN: `arn:aws:lambda:::function:GGCloudSpooler:1`

`GGConnManager`

Enruta los mensajes MQTT entre los dispositivos principales y clientes de Greengrass.

ARN: `arn:aws:lambda:::function:GGConnManager`

`GGDeviceCertificateManager`

Escucha los cambios en los AWS IoT puntos de conexión IP del núcleo y genera el certificado del lado del servidor que GGConnManager utiliza para la autenticación mutua.

ARN: `arn:aws:lambda:::function:GGDeviceCertificateManager`

`GGIPDetector`

Gestiona la detección automática de direcciones IP que permite a los dispositivos del grupo Greengrass descubrir el dispositivo central Greengrass. Este servicio no se aplica cuando se proporcionan direcciones IP manualmente.

ARN: `arn:aws:lambda:::function:GGIPDetector:1`

`GGSecretManager`

Gestiona el almacenamiento seguro de los secretos locales y el acceso mediante Lambda y conectores definidos por el usuario.

ARN: `arn:aws:lambda:::function:GGSecretManager:1`

`GGShadowService`

Administra las sombras locales para los dispositivos cliente.

ARN: `arn:aws:lambda:::function:GGShadowService`

`GGShadowSyncManager`

Sincroniza las sombras locales con la Nube de AWS para el dispositivo principal y los dispositivos de lcliente, si la propiedad `syncShadow` del dispositivo está establecida en `true`.

ARN: `arn:aws:lambda:::function:GGShadowSyncManager`

`GGStreamManager`

Procesa los flujos de datos de forma local y realiza exportaciones automáticas al Nube de AWS.

ARN: `arn:aws:lambda:::function:GGStreamManager:1`

`GGTES`

El servicio de intercambio de fichas local que recupera las credenciales de IAM definidas en el rol de grupo Greengrass que el código local utiliza para acceder a los servicios AWS.

ARN: `arn:aws:lambda:::function:GGTES`

Etiquetar los recursos de AWS IoT Greengrass

Las etiquetas pueden ayudarle a organizar y administrar sus grupos de AWS IoT Greengrass. Puede utilizar las etiquetas para asignar metadatos a grupos, implementaciones por lotes y los núcleos, dispositivos y otros recursos que se añaden a grupos. Las etiquetas también se pueden utilizar en políticas de IAM para definir acceso condicional a sus recursos de Greengrass.

> **Note**
>
> En la actualidad, no se admiten etiquetas de recursos de Greengrass con los grupos de facturación o los informes de asignación de costos de AWS IoT.

Conceptos básicos de etiquetas

Las etiquetas le permiten clasificar los recursos de AWS IoT Greengrass, por ejemplo, según su finalidad, propietario y entorno. Cuando tiene muchos recursos del mismo tipo, ya que puede identificar rápidamente un recurso en función de las etiquetas que tenga asociadas. Una etiqueta es una clave y un valor opcional, ambos definidos por el usuario. Le recomendamos que diseñe un conjunto de claves de etiqueta para cada tipo de recurso. Mediante el uso de un conjunto coherente de claves de etiquetas, podrá administrar los recursos de más fácilmente. Por ejemplo, puede definir un conjunto de etiquetas para sus grupos de modo que le resulte más fácil hacer un seguimiento de la ubicación de la fábrica de sus dispositivos principales. Para obtener más información, consulte Estrategias de etiquetado de AWS.

Compatibilidad con el etiquetado en la consola de AWS IoT

Puede crear, ver y administrar etiquetas para sus recursos de `Group` de Greengrass en la consola de AWS IoT. Antes de crear etiquetas, tenga en cuenta las restricciones de etiquetado. Para obtener más información, consulte este artículo sobre las convenciones de nomenclatura y el uso de etiquetas en la Referencia general de Amazon Web Services.

Para asignar etiquetas al crear un grupo, realice el siguiente procedimiento:

1. Puede asignar etiquetas a un grupo cuando cree uno. Seleccione Añadir nueva etiqueta en la sección Etiquetas para ver los campos de entrada del etiquetado.

Para ver y administrar etiquetas desde la página de configuración del grupo, realice el siguiente procedimiento:

Puede ver y admnistrar las etiquetas desde la página de configuración del grupo seleccionando Ver ajustes.. En la sección Etiquetas del grupo, elija Administrar etiquetas para añadir, editar o eliminar etiquetas de grupo.

Compatibilidad con el etiquetado en la API de AWS IoT Greengrass

Debe usar la API de AWS IoT Greengrass para crear, enumerar y administrar etiquetas para recursos de AWS IoT Greengrass que admiten el etiquetado. Antes de crear etiquetas, tenga en cuenta las restricciones de etiquetado. Para obtener más información, consulte este artículo sobre las convenciones de nomenclatura y el uso de etiquetas en la Referencia general de Amazon Web Services.

- Para añadir etiquetas durante la creación de recursos, defínales en la propiedad `tags` del recurso.
- Para añadir etiquetas después de crear un recurso o para actualizar valores de etiqueta, utilice la acción `TagResource`.
- Para quitar etiquetas de un recurso, utilice la acción `UntagResource`.
- Para recuperar las etiquetas que están asociadas a un recurso, utilice la acción `ListTagsForResource` u obtenga el recurso e inspeccione su propiedad `tags`.

La tabla siguiente contiene una lista de los recursos que puede etiquetar en la API de AWS IoT Greengrass y sus acciones `Create` y `Get` correspondientes.

Resource	Creación	Get
`Group`	`CreateGroup`	`GetGroup`
`ConnectorDefinition`	`CreateConnectorDefinition`	`GetConnectorDefinition`
`CoreDefinition`	`CreateCoreDefinition`	`GetCoreDefinition`
`DeviceDefinition`	`CreateDeviceDefinition`	`GetDeviceDefinition`

Resource	Creación	Get
FunctionDefinition	CreateFunctionDefinition	GetFunctionDefinition
LoggerDefinition	CreateLoggerDefinition	GetLoggerDefinition
ResourceDefinition	CreateResourceDefinition	GetResourceDefinition
SubscriptionDefinition	CreateSubscriptionDefinition	GetSubscriptionDefinition
BulkDeployment	StartBulkDeployment	GetBulkDeploymentStatus

Utilice las siguientes acciones para enumerar y administrar etiquetas para recursos que admiten el etiquetado:

- TagResource. Agrega etiquetas a un recurso. Se utiliza también para cambiar el valor del par clave-valor de la etiqueta.
- ListTagsForResource. Enumera las etiquetas de un recurso.
- UntagResource. Elimina etiquetas de un recurso.

Puede agregar o quitar etiquetas de un recurso en cualquier momento. Para cambiar el valor de una clave de etiqueta, añada una etiqueta al recurso que defina la misma clave y el nuevo valor. El valor nuevo sobrescribe el valor antiguo. Puede establecer un valor como una cadena vacía, pero no puede definir un valor como nulo.

Al eliminar un recurso, también se eliminarán las etiquetas que este tenga asociadas.

> **Note**
>
> No confunda las etiquetas de recursos a los atributos que puede asignar a objetos de AWS IoT. Aunque los núcleos de Greengrass son objetos de AWS IoT, las etiquetas de recursos

que se describen en este tema están asociados a una `CoreDefinition`, no al objeto de núcleo.

Uso de etiquetas con políticas de IAM

En las políticas de IAM, puede utilizar etiquetas de recursos para controlar los permisos y el acceso de usuarios. Por ejemplo, las políticas pueden permitir a los usuarios crear solo aquellos recursos que tienen una etiqueta específica. Las políticas también puede limitar la creación o modificación de recursos que tengan determinadas etiquetas por parte de los usuarios. Puede etiquetar recursos durante la creación (denominados etiqueta durante la creación) para no tener que ejecutar scripts de etiquetado personalizados más tarde. Cuando se lanzan nuevos entornos con etiquetas, los permisos de IAM correspondientes se aplican automáticamente.

Las siguientes claves y valores de contexto de condición se pueden utilizar en el elemento `Condition` (también denominado bloque `Condition`) de la política.

`greengrass:ResourceTag/`*`tag-key`*`: `*`tag-value`*

Permita o deniegue acciones de los usuarios en recursos con etiquetas específicas.

`aws:RequestTag/`*`tag-key`*`: `*`tag-value`*

Requiera el uso de una etiqueta específica (o no utilizada) al realizar solicitudes de la API para crear o modificar etiquetas en un recurso etiquetable.

`aws:TagKeys: [`*`tag-key`*`, ...]`

Requiera el uso de un conjunto de claves de etiqueta específico (o no utilizado) al realizar una solicitud a la API para crear o modificar un recurso etiquetable.

Los valores y las claves de contexto de condición solo se pueden utilizar en acciones de AWS IoT Greengrass que actúan sobre un recurso etiquetable. Estas acciones toman el recurso como parámetro requerido. Por ejemplo, puede establecer acceso condicional en el `GetGroupVersion`. No puede establecer acceso condicional en `AssociateServiceRoleToAccount` dado que no se hace referencia a ningún recurso etiquetable (por ejemplo, grupo, definición de núcleo o definición de dispositivo) en la solicitud.

Para obtener más información, consulte Control de acceso mediante etiquetas y referencia de políticas JSON de IAM en la Guía del usuario de IAM. La referencia de política JSON incluye la

sintaxis, descripciones y ejemplos detallados de los elementos, variables y lógica de evaluación de las políticas JSON en IAM.

Ejemplos de políticas de IAM

La política del ejemplo siguiente aplica permisos basados en etiquetas que limitan a un usuario beta solo a acciones en recursos beta.

- La primera instrucción permite a un usuario de IAM actuar sobre recursos que tienen solo la etiqueta env=beta.
- La segunda instrucción evita que un usuario de IAM quite la etiqueta env=beta de los recursos. Esto protege al usuario de eliminar su propio acceso.

> **Note**
>
> Si utiliza etiquetas para controlar el acceso a recursos, también debe administrar los permisos que permiten a los usuarios añadir o quitar etiquetas de estos mismos recursos. De lo contrario, en algunos casos, sería posible para que los usuarios eludan sus restricciones y obtengan acceso a un recurso modificando sus etiquetas.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "greengrass:ResourceTag/env": "beta"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": "greengrass:UntagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:ResourceTag/env": "beta"
                }
            }
        }
    ]
}
```

Para permitir a los usuarios etiquetar durante la creación, debe proporcionarles los permisos adecuados. La siguiente política de ejemplo incluye la condición `"aws:RequestTag/env": "beta"` en las acciones `greengrass:TagResource` y `greengrass:CreateGroup`, que permiten a los usuarios crear un grupo solo si etiquetan el grupo con env=beta. Esto fuerza a los usuarios de manera eficiente a etiquetar nuevos grupos.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "beta"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "greengrass:CreateGroup",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "beta"
                }
            }
        }
    ]
}
```

El siguiente fragmento muestra cómo puede especificar varios valores de etiqueta para una clave de etiqueta encerrándola en una lista:

```
"StringEquals" : {
    "greengrass:ResourceTag/env" : ["dev", "test"]
}
```

Véase también

- Etiquetado de recursos de AWS en Referencia general de Amazon Web Services.

Compatibilidad con AWS CloudFormation para AWS IoT Greengrass

AWS CloudFormation es un servicio que puede ayudarle a crear, administrar y replicar sus recursos de AWS. Puede utilizar plantillas de AWS CloudFormation para definir grupos de AWS IoT Greengrass y los dispositivos del cliente, suscripciones y otros componentes que desee implementar. Para ver un ejemplo, consulte the section called “Ejemplo de plantilla de de ”.

Los recursos y la infraestructura que genere desde una plantilla se denomina pila. Puede definir todos los recursos en una plantilla o hacer referencia a recursos de otras pilas. Para obtener más información acerca de las plantillas y características de AWS CloudFormation, consulte ¿Qué es AWS CloudFormation? en la Guía del usuario de AWS CloudFormation..

Creación de recursos de

Las plantillas de AWS CloudFormation son documentos JSON o YAML que describen las propiedades y relaciones de los recursos de AWS. Se admiten los siguientes recursos de AWS IoT Greengrass:

- Grupos
- Núcleos
- Dispositivos cliente (dispositivos)
- Funciones de Lambda
- Connectors
- Recursos (local, aprendizaje automático y secreto)
- Suscripciones
- Registradores (configuraciones de registro)

En las plantillas de AWS CloudFormation, la estructura y la sintaxis de recursos de Greengrass se basan en la API de AWS IoT Greengrass. Por ejemplo, la plantilla de ejemplo asocia una `DeviceDefinition` de nivel superior con una `DeviceDefinitionVersion` que contiene un dispositivo de cliente individual. Para obtener más información, consulte the section called “Información general sobre el modelo de objetos de grupo”.

La Referencia de tipos de recursos de AWS IoT Greengrass en la Guía del usuario de AWS CloudFormation describe los recursos de Greengrass que puede administrar con AWS CloudFormation. Cuando se utilizan plantillas de AWS CloudFormation para crear recursos de Greengrass, recomendamos que solo las administre desde AWS CloudFormation. Por ejemplo, debería actualizar la plantilla si desea añadir, cambiar o quitar un dispositivo (en lugar de utilizar la API de AWS IoT Greengrass o consola AWS IoT). Esto le permite utilizar la restauración y otras características de gestión de cambios de AWS CloudFormation. Para obtener más información sobre el uso de AWS CloudFormation para crear y administrar los recursos y pilas, consulte Uso de pilas en la Guía del usuario de AWS CloudFormation.

Para obtener una explicación detallada en la que se muestre cómo crear e implementar recursos de AWS IoT Greengrass en una plantilla de AWS CloudFormation, consulte Automatizar una configuración de AWS IoT Greengrass con AWS CloudFormation en el blog oficial del Internet de las cosas de AWS.

Implementación de recursos

Después de crear una pila de AWS CloudFormation que contiene la versión de grupo, puede utilizar la AWS CLI o la consola AWS IoT para implementarla.

> **Note**
>
> Para implementar un grupo, debe tener un rol de servicio de Greengrass asociado a su Cuenta de AWS. El rol de servicio permite a AWS IoT Greengrass acceder a sus recursos en AWS Lambda y otros servicios de AWS. Este rol debe existir si ya ha implementado un grupo de Greengrass en la región de Región de AWS actual. Para obtener más información, consulte the section called “Rol de servicio de Greengrass”.

Para implementar el grupo (AWS CLI)

- Ejecute el comando .create-deployment

```
aws greengrass create-deployment --group-id GroupId --group-version-
id GroupVersionId --deployment-type NewDeployment
```

> **Note**
>
> La instrucción `CommandToDeployGroup` en la plantilla de ejemplo muestra cómo generar la salida del comando con los ID de su grupo y versión de grupo al crear una pila.

Para implementar el grupo (consola)

1. En el panel de navegación de la consola AWS IoT, en Administrar, expanda los dispositivos Greengrass y, a continuación, elija Grupos (V1).
2. Elija su grupo.
3. En la página de configuración de grupo, elija Implementar.

Ejemplo de plantilla de de

La siguiente plantilla de ejemplo crea un grupo de Greengrass que contiene un núcleo, dispositivo del cliente, función, registrador, suscripción y dos recursos. Para ello, la plantilla sigue el modelo de objeto de la API de AWS IoT Greengrass. Por ejemplo, los dispositivos del cliente que desea añadir al grupo están contenidos en un recurso de `DeviceDefinitionVersion`, que está asociado a un recurso de `DeviceDefinition`. Para añadir los dispositivos al grupo, la versión del grupo hace referencia al ARN del `DeviceDefinitionVersion`.

La plantilla incluye parámetros que le permiten especificar los ARN de certificado para el núcleo y dispositivo y el ARN de versión de la función de Lambda origen (que es un recurso de AWS Lambda). Utiliza las funciones intrínsecas `Ref` y `GetAtt` para hacer referencia a ID, ARN y otros atributos necesarios para crear recursos de Greengrass.

La plantilla define además dos dispositivos de AWS IoT (objetos), que representan el núcleo y dispositivo del cliente que se añaden al grupo de Greengrass.

Después de crear la pila con los recursos de Greengrass, puede utilizar la AWS CLI o la consola AWS IoT para implementar el grupo.

> **Note**
>
> La instrucción `CommandToDeployGroup` en el ejemplo muestra cómo generar la salida de un comando de CLI create-deployment completo que puede utilizar para implementar el grupo.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "AWS IoT Greengrass example template that creates a group version
 with a core, device, function, logger, subscription, and resources.",
    "Parameters": {
        "CoreCertificateArn": {
            "Type": "String"
        },
        "DeviceCertificateArn": {
            "Type": "String"
        },
        "LambdaVersionArn": {
            "Type": "String"
        }
    },
    "Resources": {
        "TestCore1": {
            "Type": "AWS::IoT::Thing",
            "Properties": {
                "ThingName": "TestCore1"
            }
        },
        "TestCoreDefinition": {
            "Type": "AWS::Greengrass::CoreDefinition",
            "Properties": {
                "Name": "DemoTestCoreDefinition"
            }
        },
        "TestCoreDefinitionVersion": {
            "Type": "AWS::Greengrass::CoreDefinitionVersion",
            "Properties": {
                "CoreDefinitionId": {
                    "Ref": "TestCoreDefinition"
```

```
                },
                "Cores": [
                    {
                        "Id": "TestCore1",
                        "CertificateArn": {
                            "Ref": "CoreCertificateArn"
                        },
                        "SyncShadow": "false",
                        "ThingArn": {
                            "Fn::Join": [
                                ":",
                                [
                                    "arn:aws:iot",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    "thing/TestCore1"
                                ]
                            ]
                        }
                    }
                ]
            }
        },
        "TestClientDevice1": {
            "Type": "AWS::IoT::Thing",
            "Properties": {
                "ThingName": "TestClientDevice1"
            }
        },
        "TestDeviceDefinition": {
            "Type": "AWS::Greengrass::DeviceDefinition",
            "Properties": {
                "Name": "DemoTestDeviceDefinition"
            }
        },
        "TestDeviceDefinitionVersion": {
            "Type": "AWS::Greengrass::DeviceDefinitionVersion",
            "Properties": {
                "DeviceDefinitionId": {
                    "Fn::GetAtt": [
```

```
                    "TestDeviceDefinition",
                    "Id"
                ]
            },
            "Devices": [
                {
                    "Id": "TestClientDevice1",
                    "CertificateArn": {
                        "Ref": "DeviceCertificateArn"
                    },
                    "SyncShadow": "true",
                    "ThingArn": {
                        "Fn::Join": [
                            ":",
                            [
                                "arn:aws:iot",
                                {
                                    "Ref": "AWS::Region"
                                },
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                "thing/TestClientDevice1"
                            ]
                        ]
                    }
                }
            ]
        }
    },
    "TestFunctionDefinition": {
        "Type": "AWS::Greengrass::FunctionDefinition",
        "Properties": {
            "Name": "DemoTestFunctionDefinition"
        }
    },
    "TestFunctionDefinitionVersion": {
        "Type": "AWS::Greengrass::FunctionDefinitionVersion",
        "Properties": {
            "FunctionDefinitionId": {
                "Fn::GetAtt": [
                    "TestFunctionDefinition",
                    "Id"
                ]
```

```
                },
                "DefaultConfig": {
                    "Execution": {
                        "IsolationMode": "GreengrassContainer"
                    }
                },
                "Functions": [
                    {
                        "Id": "TestLambda1",
                        "FunctionArn": {
                            "Ref": "LambdaVersionArn"
                        },
                        "FunctionConfiguration": {
                            "Pinned": "true",
                            "Executable": "run.exe",
                            "ExecArgs": "argument1",
                            "MemorySize": "512",
                            "Timeout": "2000",
                            "EncodingType": "binary",
                            "Environment": {
                                "Variables": {
                                    "variable1": "value1"
                                },
                                "ResourceAccessPolicies": [
                                    {
                                        "ResourceId": "ResourceId1",
                                        "Permission": "ro"
                                    },
                                    {
                                        "ResourceId": "ResourceId2",
                                        "Permission": "rw"
                                    }
                                ],
                                "AccessSysfs": "false",
                                "Execution": {
                                    "IsolationMode": "GreengrassContainer",
                                    "RunAs": {
                                        "Uid": "1",
                                        "Gid": "10"
                                    }
                                }
                            }
                        }
                    }
```

```
            ]
        }
    },
    "TestLoggerDefinition": {
        "Type": "AWS::Greengrass::LoggerDefinition",
        "Properties": {
            "Name": "DemoTestLoggerDefinition"
        }
    },
    "TestLoggerDefinitionVersion": {
        "Type": "AWS::Greengrass::LoggerDefinitionVersion",
        "Properties": {
            "LoggerDefinitionId": {
                "Ref": "TestLoggerDefinition"
            },
            "Loggers": [
                {
                    "Id": "TestLogger1",
                    "Type": "AWSCloudWatch",
                    "Component": "GreengrassSystem",
                    "Level": "INFO"
                }
            ]
        }
    },
    "TestResourceDefinition": {
        "Type": "AWS::Greengrass::ResourceDefinition",
        "Properties": {
            "Name": "DemoTestResourceDefinition"
        }
    },
    "TestResourceDefinitionVersion": {
        "Type": "AWS::Greengrass::ResourceDefinitionVersion",
        "Properties": {
            "ResourceDefinitionId": {
                "Ref": "TestResourceDefinition"
            },
            "Resources": [
                {
                    "Id": "ResourceId1",
                    "Name": "LocalDeviceResource",
                    "ResourceDataContainer": {
                        "LocalDeviceResourceData": {
                            "SourcePath": "/dev/TestSourcePath1",
```

```
                            "GroupOwnerSetting": {
                                "AutoAddGroupOwner": "false",
                                "GroupOwner": "TestOwner"
                            }
                        }
                    }
                },
                {
                    "Id": "ResourceId2",
                    "Name": "LocalVolumeResourceData",
                    "ResourceDataContainer": {
                        "LocalVolumeResourceData": {
                            "SourcePath": "/dev/TestSourcePath2",
                            "DestinationPath": "/volumes/TestDestinationPath2",
                            "GroupOwnerSetting": {
                                "AutoAddGroupOwner": "false",
                                "GroupOwner": "TestOwner"
                            }
                        }
                    }
                }
            ]
        }
    },
    "TestSubscriptionDefinition": {
        "Type": "AWS::Greengrass::SubscriptionDefinition",
        "Properties": {
            "Name": "DemoTestSubscriptionDefinition"
        }
    },
    "TestSubscriptionDefinitionVersion": {
        "Type": "AWS::Greengrass::SubscriptionDefinitionVersion",
        "Properties": {
            "SubscriptionDefinitionId": {
                "Ref": "TestSubscriptionDefinition"
            },
            "Subscriptions": [
                {
                    "Id": "TestSubscription1",
                    "Source": {
                        "Fn::Join": [
                            ":",
                            [
                                "arn:aws:iot",
```

```
                                {
                                    "Ref": "AWS::Region"
                                },
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                "thing/TestClientDevice1"
                            ]
                        ]
                    },
                    "Subject": "TestSubjectUpdated",
                    "Target": {
                        "Ref": "LambdaVersionArn"
                    }
                }
            ]
        }
    },
    "TestGroup": {
        "Type": "AWS::Greengrass::Group",
        "Properties": {
            "Name": "DemoTestGroupNewName",
            "RoleArn": {
                "Fn::Join": [
                    ":",
                    [
                        "arn:aws:iam:",
                        {
                            "Ref": "AWS::AccountId"
                        },
                        "role/TestUser"
                    ]
                ]
            },
            "InitialVersion": {
                "CoreDefinitionVersionArn": {
                    "Ref": "TestCoreDefinitionVersion"
                },
                "DeviceDefinitionVersionArn": {
                    "Ref": "TestDeviceDefinitionVersion"
                },
                "FunctionDefinitionVersionArn": {
                    "Ref": "TestFunctionDefinitionVersion"
                },
```

```
                    "SubscriptionDefinitionVersionArn": {
                        "Ref": "TestSubscriptionDefinitionVersion"
                    },
                    "LoggerDefinitionVersionArn": {
                        "Ref": "TestLoggerDefinitionVersion"
                    },
                    "ResourceDefinitionVersionArn": {
                        "Ref": "TestResourceDefinitionVersion"
                    }
                },
                "Tags": {
                    "KeyName0": "value",
                    "KeyName1": "value",
                    "KeyName2": "value"
                }
            }
        }
    },
    "Outputs": {
        "CommandToDeployGroup": {
            "Value": {
                "Fn::Join": [
                    " ",
                    [
                        "groupVersion=$(cut -d'/' -f6 <<<",
                        {
                            "Fn::GetAtt": [
                                "TestGroup",
                                "LatestVersionArn"
                            ]
                        },
                        ");",
                        "aws --region",
                        {
                            "Ref": "AWS::Region"
                        },
                        "greengrass create-deployment --group-id",
                        {
                            "Ref": "TestGroup"
                        },
                        "--deployment-type NewDeployment --group-version-id",
                        "$groupVersion"
                    ]
                ]
```

```
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  AWS IoT Greengrass example template that creates a group version with a core,
  device, function, logger, subscription, and resources.
Parameters:
  CoreCertificateArn:
    Type: String
  DeviceCertificateArn:
    Type: String
  LambdaVersionArn:
    Type: String
Resources:
  TestCore1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestCore1
  TestCoreDefinition:
    Type: 'AWS::Greengrass::CoreDefinition'
    Properties:
      Name: DemoTestCoreDefinition
  TestCoreDefinitionVersion:
    Type: 'AWS::Greengrass::CoreDefinitionVersion'
    Properties:
      CoreDefinitionId: !Ref TestCoreDefinition
      Cores:
        - Id: TestCore1
          CertificateArn: !Ref CoreCertificateArn
          SyncShadow: 'false'
          ThingArn: !Join
            - ':'
            - - 'arn:aws:iot'
              - !Ref 'AWS::Region'
              - !Ref 'AWS::AccountId'
              - thing/TestCore1
  TestClientDevice1:
    Type: 'AWS::IoT::Thing'
```

```
  Properties:
    ThingName: TestClientDevice1
TestDeviceDefinition:
  Type: 'AWS::Greengrass::DeviceDefinition'
  Properties:
    Name: DemoTestDeviceDefinition
TestDeviceDefinitionVersion:
  Type: 'AWS::Greengrass::DeviceDefinitionVersion'
  Properties:
    DeviceDefinitionId: !GetAtt
      - TestDeviceDefinition
      - Id
    Devices:
      - Id: TestClientDevice1
        CertificateArn: !Ref DeviceCertificateArn
        SyncShadow: 'true'
        ThingArn: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestClientDevice1
TestFunctionDefinition:
  Type: 'AWS::Greengrass::FunctionDefinition'
  Properties:
    Name: DemoTestFunctionDefinition
TestFunctionDefinitionVersion:
  Type: 'AWS::Greengrass::FunctionDefinitionVersion'
  Properties:
    FunctionDefinitionId: !GetAtt
      - TestFunctionDefinition
      - Id
    DefaultConfig:
      Execution:
        IsolationMode: GreengrassContainer
    Functions:
      - Id: TestLambda1
        FunctionArn: !Ref LambdaVersionArn
        FunctionConfiguration:
          Pinned: 'true'
          Executable: run.exe
          ExecArgs: argument1
          MemorySize: '512'
          Timeout: '2000'
```

```
            EncodingType: binary
            Environment:
              Variables:
                variable1: value1
              ResourceAccessPolicies:
                - ResourceId: ResourceId1
                  Permission: ro
                - ResourceId: ResourceId2
                  Permission: rw
              AccessSysfs: 'false'
              Execution:
                IsolationMode: GreengrassContainer
                RunAs:
                  Uid: '1'
                  Gid: '10'
  TestLoggerDefinition:
    Type: 'AWS::Greengrass::LoggerDefinition'
    Properties:
      Name: DemoTestLoggerDefinition
  TestLoggerDefinitionVersion:
    Type: 'AWS::Greengrass::LoggerDefinitionVersion'
    Properties:
      LoggerDefinitionId: !Ref TestLoggerDefinition
      Loggers:
        - Id: TestLogger1
          Type: AWSCloudWatch
          Component: GreengrassSystem
          Level: INFO
  TestResourceDefinition:
    Type: 'AWS::Greengrass::ResourceDefinition'
    Properties:
      Name: DemoTestResourceDefinition
  TestResourceDefinitionVersion:
    Type: 'AWS::Greengrass::ResourceDefinitionVersion'
    Properties:
      ResourceDefinitionId: !Ref TestResourceDefinition
      Resources:
        - Id: ResourceId1
          Name: LocalDeviceResource
          ResourceDataContainer:
            LocalDeviceResourceData:
              SourcePath: /dev/TestSourcePath1
              GroupOwnerSetting:
                AutoAddGroupOwner: 'false'
```

```
              GroupOwner: TestOwner
      - Id: ResourceId2
        Name: LocalVolumeResourceData
        ResourceDataContainer:
          LocalVolumeResourceData:
            SourcePath: /dev/TestSourcePath2
            DestinationPath: /volumes/TestDestinationPath2
            GroupOwnerSetting:
              AutoAddGroupOwner: 'false'
              GroupOwner: TestOwner
TestSubscriptionDefinition:
  Type: 'AWS::Greengrass::SubscriptionDefinition'
  Properties:
    Name: DemoTestSubscriptionDefinition
TestSubscriptionDefinitionVersion:
  Type: 'AWS::Greengrass::SubscriptionDefinitionVersion'
  Properties:
    SubscriptionDefinitionId: !Ref TestSubscriptionDefinition
    Subscriptions:
      - Id: TestSubscription1
        Source: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestClientDevice1
        Subject: TestSubjectUpdated
        Target: !Ref LambdaVersionArn
TestGroup:
  Type: 'AWS::Greengrass::Group'
  Properties:
    Name: DemoTestGroupNewName
    RoleArn: !Join
      - ':'
      - - 'arn:aws:iam:'
        - !Ref 'AWS::AccountId'
        - role/TestUser
    InitialVersion:
      CoreDefinitionVersionArn: !Ref TestCoreDefinitionVersion
      DeviceDefinitionVersionArn: !Ref TestDeviceDefinitionVersion
      FunctionDefinitionVersionArn: !Ref TestFunctionDefinitionVersion
      SubscriptionDefinitionVersionArn: !Ref TestSubscriptionDefinitionVersion
      LoggerDefinitionVersionArn: !Ref TestLoggerDefinitionVersion
      ResourceDefinitionVersionArn: !Ref TestResourceDefinitionVersion
```

```
      Tags:
        KeyName0: value
        KeyName1: value
        KeyName2: value
Outputs:
  CommandToDeployGroup:
    Value: !Join
      - ' '
      - - groupVersion=$(cut -d'/' -f6 <<<
        - !GetAtt
          - TestGroup
          - LatestVersionArn
        - );
        - aws --region
        - !Ref 'AWS::Region'
        - greengrass create-deployment --group-id
        - !Ref TestGroup
        - '--deployment-type NewDeployment --group-version-id'
        - $groupVersion
```

Región de AWS admitidas

En la actualidad, puede crear y administrar recursos de AWS IoT Greengrass únicamente en las siguientes Región de AWS:

- US East (Ohio)
- Este de EE. UU. (Norte de Virginia)
- Oeste de EE. UU. (Oregón)
- Asia-Pacífico (Bombay)
- Asia-Pacífico (Seúl)
- Asia-Pacífico (Singapur)
- Asia-Pacífico (Sídney)
- Asia-Pacífico (Tokio)
- China (Pekín)
- Europa (Fráncfort)
- Europa (Irlanda)
- Europa (Londres)

- AWS GovCloud (US-Oeste)

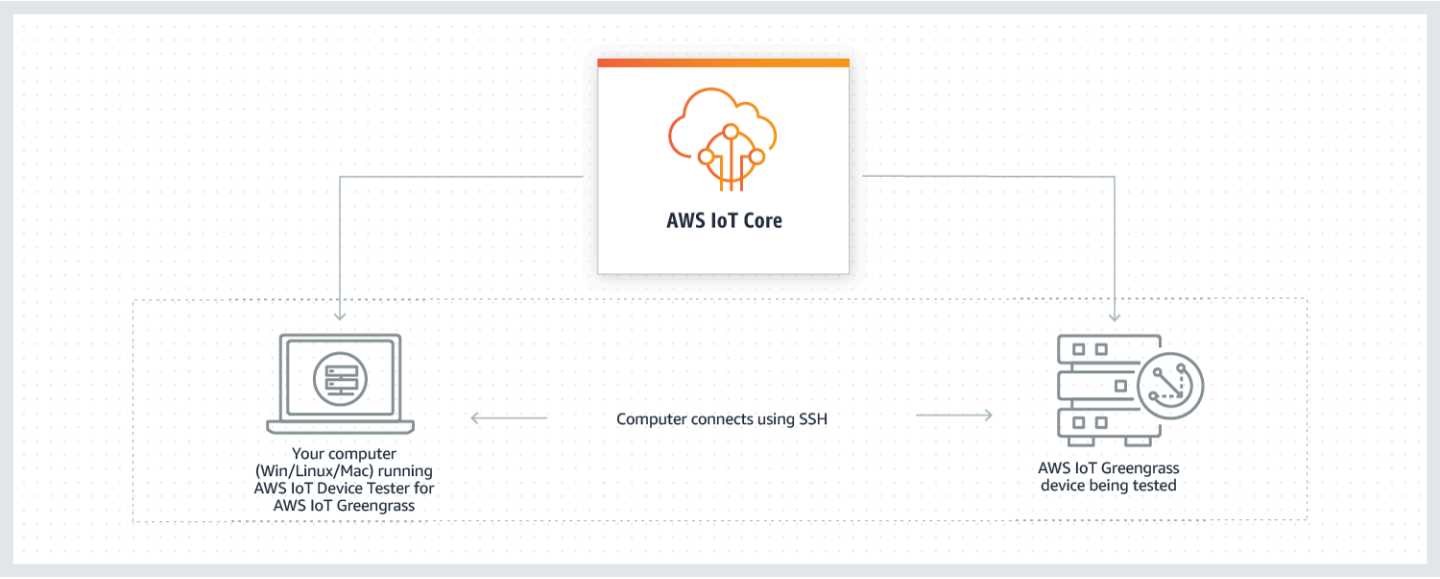
Uso AWS IoT del comprobador de dispositivos para V1 AWS IoT Greengrass

AWS IoT Device Tester (IDT) es un marco de pruebas descargable que le permite validar dispositivos de IoT. Dado que AWS IoT Greengrass Version 1 ha pasado al modo de mantenimiento, IDT for ya AWS IoT Greengrass V1 no genera informes de cualificación firmados. Ya no podrá incluir nuevos AWS IoT Greengrass V1 dispositivos en el catálogo de dispositivos a través del AWS Partner Programa de calificación de AWS dispositivos. Sin embargo, puede seguir utilizando IDT AWS IoT Greengrass V1 para probar sus dispositivos Greengrass V1. Le recomendamos que utilice IDT para AWS IoT Greengrass V2 para calificar y publicar los dispositivos de Greengrass en el Catálogo de dispositivos de AWS Partner.

IDT for AWS IoT Greengrass se ejecuta en el ordenador anfitrión (Windows, macOS o Linux) conectado al dispositivo que se va a probar. Ejecuta pruebas y agrega resultados. También proporciona una interfaz de línea de comandos para administrar el proceso de pruebas.

AWS IoT Greengrass paquete de cualificación

Utilice IDT AWS IoT Greengrass para comprobar que el software AWS IoT Greengrass principal se ejecuta en su hardware y puede comunicarse con el Nube de AWS. También realiza end-to-end pruebas con AWS IoT Core. Por ejemplo, verifica que su dispositivo pueda enviar y recibir mensajes MQTT y procesarlos correctamente.



AWS IoT Device Tester para AWS IoT Greengrass organizar las pruebas utilizando los conceptos de conjuntos de pruebas y grupos de pruebas.

- Un conjunto de pruebas es el conjunto de grupos de pruebas que se utiliza para verificar que un dispositivo funciona con versiones particulares de AWS IoT Greengrass.
- Un grupo de pruebas es el conjunto de pruebas individuales relacionadas con una característica concreta, como implementaciones de grupos de Greengrass y mensajería MQTT.

Para obtener más información, consulte Utilice IDT para ejecutar el conjunto de cualificación de AWS IoT Greengrass.

Compatibilidad con los conjuntos de prueba

A partir de la versión 4.0.0 de IDT, IDT for AWS IoT Greengrass combina una configuración y un formato de resultados estandarizados con un entorno de conjuntos de pruebas que le permite desarrollar conjuntos de pruebas personalizados para sus dispositivos y el software de sus dispositivos. Puede añadir pruebas personalizadas para su propia validación interna o proporcionárselas a sus clientes para la verificación de los dispositivos.

La forma en que un escritor de pruebas configura un conjunto de pruebas personalizado determina las configuraciones de configuración necesarias para ejecutar conjuntos de pruebas personalizados. Para obtener más información, consulte Utilice IDT para desarrollar y ejecutar sus propios conjuntos de pruebas.

Versiones de AWS IoT Device Tester compatibles con AWS IoT Greengrass V1.

Dado que AWS IoT Greengrass Version 1 ha pasado al modo de mantenimiento, IDT para AWS IoT Greengrass V1 ya no genera informes de cualificación firmados. Le recomendamos que utilice IDT para AWS IoT Greengrass V2..

Para obtener más información acerca de IDT para AWS IoT Greengrass V2, consulte Uso de Device Tester de AWS IoT para AWS IoT Greengrass V2 en la Guía del usuario de AWS IoT Greengrass V2.

> **Note**
>
> Recibirá una notificación cuando inicie una ejecución de prueba si IDT para AWS IoT Greengrass no es compatible con la versión de AWS IoT Greengrass que está utilizando.

Al descargar el software, acepta el Acuerdo de licencia de AWS IoT Device Tester.

Versiones no compatibles de IDT para AWS IoT Greengrass

En este tema, se muestran las versiones no compatibles de IDT para AWS IoT Greengrass. Las versiones que no son compatibles no reciben actualizaciones ni correcciones de errores. Para obtener más información, consulte the section called “Política de compatibilidad de AWS IoT Device Tester para AWS IoT Greengrass V1”.

IDT versión 4.4.1 para AWS IoT Greengrass versiones 1.11.6, 1.10.5

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan el software AWS IoT Greengrass Core versiones 1.11.6 y v 1.10.5.
- Contiene correcciones de errores menores.

Versión del conjunto de pruebas:

`GGQ_1.3.1`

- Publicado el 20/12/2020

IDT versión 4.1.0 para versiones de AWS IoT Greengrass 1.11.4, 1.10.4

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan el software AWS IoT Greengrass Core versiones 1.11.4 y 1.10.4.
- Soluciona un problema que provocaba que los registros que se muestran durante una ejecución de prueba utilizaran etiquetas redundantes.

Versión del conjunto de pruebas:

`GGQ_1.3.0`

- Publicado el 23/06/2021
- Añade reintentos para las llamadas de API a Lambda, IAM y AWS STS para mejorar la gestión de los problemas de limitación o de servidor.

- Añade compatibilidad con Python 3.8 a los casos de prueba de ML y Docker.

IDT versión 4.0.2 para AWS IoT Greengrass, versiones 1.11.1, 1.11.0, 1.10.3

Notas de la versión:

- Solucionaba un problema que provocaba que IDT ocultara los errores de Hardware Security Integration (HSI).
- Le permite desarrollar y ejecutar sus conjuntos de pruebas personalizados con AWS IoT Device Tester para AWS IoT Greengrass. Para obtener más información, consulte Utilice IDT para desarrollar y ejecutar sus propios conjuntos de pruebas.
- Proporciona aplicaciones IDT con firma de código para macOS y Windows. En macOS, si aparece un mensaje de advertencia de seguridad, es posible que tenga que conceder una excepción de seguridad para IDT. Para obtener más información, consulte Excepción de seguridad en macOS.

> **Note**
>
> AWS IoT Greengrass no proporciona un Dockerfile o una imagen de Docker para la versión 1.11.1 del software AWS IoT Greengrass Core. Para comprobar si su dispositivo cumple los requisitos de Docker, utilice una versión anterior del software AWS IoT Greengrass Core.

IDT versión 3.2.0 para AWS IoT Greengrass, versiones 1.11.0, 1.10.1, 1.10.0

Notas de la versión:

- De forma predeterminada, IDT solo ejecuta las pruebas obligatorias para la calificación. Para poder disfrutar de funciones adicionales, puede modificar el archivo `device.json`.
- Se agregó un número de puerto `device.json` que puede configurar para las conexiones SSH.
- Docker solo admite el administrador de secuencias y machine learning (ML) sin organización en contenedores. Los dispositivos Container, Docker y Hardware Security Integration (HSI) no están disponibles para los dispositivos Docker.
- Fusionamos `device-ml.json` y `device-hsm.json` y formamos `device.json`.

IDT 3.1.3 para versiones de AWS IoT Greengrass 1.9.x, 1.8.x y 1.7.x

Notas de la versión:

- Se ha agregado compatibilidad para la cualificación de características ML para AWS IoT Greengrass v1.10.x y v1.9.x. Ahora puede usar IDT para validar que los dispositivos pueden realizar inferencia de ML localmente con modelos almacenados y entrenados en la nube.
- Se ha agregado `--stop-on-first-failure` para el comando `run-suite`. Puede utilizar esta opción para configurar IDT para que deje de funcionar en el primer error. Recomendamos usar esta opción durante la etapa de depuración en el nivel de grupos de prueba.
- Se ha agregado una comprobación de deriva de reloj para las pruebas MQTT para asegurarse de que el dispositivo bajo prueba utilice la hora correcta del sistema. El tiempo utilizado debe estar dentro de un rango de tiempo aceptable.
- Se ha agregado `--update-idt` para el comando `run-suite`. Puede utilizar esta opción para establecer la respuesta del mensaje para actualizar IDT.
- Se ha agregado `--update-managed-policy` para el comando `run-suite`. Puede utilizar esta opción para establecer la respuesta para el mensaje de actualización de la política administrada.
- Se ha añadido una corrección de errores para las actualizaciones automáticas de las versiones del conjunto de pruebas de IDT. La corrección garantiza que IDT pueda ejecutar los últimos conjuntos de pruebas disponibles para su versión de AWS IoT Greengrass.

IDT v3.0.1 para AWS IoT Greengrass

Notas de la versión:

- Se ha agregado compatibilidad con AWS IoT Greengrass v1.10.1.
- Actualizaciones automáticas de las versiones del conjunto de pruebas de IDT. IDT puede descargar los conjuntos de pruebas más recientes disponibles para su versión de AWS IoT Greengrass. Con esta característica:
  - Los conjuntos de pruebas se versionan utilizando un formato *`major`*`.`*`minor`*`.`*`patch`*. La versión inicial del conjunto de pruebas es `GGQ_1.0.0`.
  - Puede descargar nuevos conjuntos de pruebas de forma interactiva en la interfaz de línea de comandos o establecer el indicador `upgrade-test-suite` cuando inicie IDT.

Para obtener más información, consulte the section called “Versiones del conjunto de pruebas”.

- `list-supported-products` añadido. Puede utilizar este comando para mostrar las versiones del conjunto de pruebas de AWS IoT Greengrass que son compatibles con la versión instalada de IDT.
- `list-test-cases` añadido. Puede utilizar este comando para mostrar los casos de prueba que están disponibles en un grupo de pruebas.
- Se ha agregado `test-id` para el comando `run-suite`. Puede utilizar esta opción para ejecutar casos de prueba individuales en un grupo de pruebas.

IDT v2.3.0 para AWS IoT Greengrass v1.10, v1.9.x y v1.8.x

Al realizar pruebas en un dispositivo físico, se admiten las versiones 1.10, 1.9.x y 1.8.x de AWS IoT Greengrass.

Al realizar pruebas en un contenedor de Docker, se admiten las versiones 1.10 y 1.9.x de AWS IoT Greengrass.

Notas de la versión:

- Se agregó compatibilidad con the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”. Ahora puede utilizar IDT para cualificar y validar que los dispositivos puedan ejecutar AWS IoT Greengrass en un contenedor de Docker.
- Se ha añadido una AWSpolítica administrada de (`AWSIoTDeviceTesterForGreengrassFullAccess`) que define los permisos necesarios para ejecutar AWS IoT Device Tester. Si las nuevas versiones requieren permisos adicionales, AWS los añade a esta política administrada para que no tenga que actualizar los permisos de IAM.
- Se han introducido comprobaciones para validar que el entorno (por ejemplo, la conectividad de los dispositivos y la conectividad de Internet) esté configurado correctamente antes de ejecutar los casos de prueba.
- Se ha mejorado el comprobador de dependencias de Greengrass en IDT para hacerlo más flexible durante la comprobación de libc en los dispositivos.

IDT v2.2.0 para AWS IoT Greengrass v1.10, v1.9.x y v1.8.x

Notas de la versión:

- Se ha agregado compatibilidad con AWS IoT Greengrass v1.10.
- Se ha agregado compatibilidad con el conector de implementación de la aplicación de Greengrass Docker .
- Se agregó soporte para el administrador de secuencias de AWS IoT Greengrass.
- Se agregó soporte para AWS IoT Greengrass en la región de China (Pekín)

IDT v2.1.0 para AWS IoT Greengrass v1.9.x, v1.8.x y v1.7.x

Notas de la versión:

- Se ha agregado compatibilidad con AWS IoT Greengrass v1.9.4.
- Se ha agregado compatibilidad para dispositivos Linux-ARMv6l.

IDT v2.0.0 para AWS IoT Greengrass v1.9.3, v1.9.2, v.1.9.1, v1.9.0, v1.8.4, v1.8.3 y v1.8.2

Notas de la versión:

- Se ha eliminado la dependencia de Python para el dispositivo en proceso de prueba.
- El tiempo de ejecución del conjunto de pruebas se redujo en más del 50 por ciento, lo que agiliza el proceso de cualificación.
- El tamaño ejecutable se ha reducido en más del 50 por ciento, lo que hace que la descarga y la instalación sean más rápidas.
- Se ha mejorado la compatibilidad del multiplicador de tiempo de espera para todos los casos de prueba.
- Mensajes posteriores al diagnóstico mejorados para solucionar los errores con mayor rapidez.
- Se ha actualizado la plantilla de política de permisos necesaria para ejecutar IDT.
- Se ha agregado compatibilidad con AWS IoT Greengrass v1.9.3.

IDT v1.3.3 para AWS IoT Greengrass v1.9.2, v1.9.1, v1.9.0, v1.8.3 y v1.8.2

Notas de la versión:

- Se ha agregado compatibilidad con Greengrass v1.9.2 y v1.8.3.

- Se ha añadido soporte para Greengrass OpenWrt.
- Se ha añadido el inicio de sesión del dispositivo con nombre de usuario y contraseña SSH.
- Se agregó una corrección de error de prueba nativa para la plataforma OpenWrt -ARMv7L.

IDT v1.2 para AWS IoT Greengrass v1.8.1

Notas de la versión:

- Se ha añadido un multiplicador de tiempo de espera configurable para abordar y solucionar los problemas de tiempo de espera (por ejemplo, conexiones de ancho de banda bajo).

IDT v1.1 para AWS IoT Greengrass v1.8.0

Notas de la versión:

- Se ha añadido compatibilidad para la integración de seguridad de hardware (HSI) de AWS IoT Greengrass.
- Se ha añadido compatibilidad para contenedor y sin contenedor de AWS IoT Greengrass.
- Se ha añadido la creación automatizada del rol de servicio de AWS IoT Greengrass.
- Se ha mejorado la limpieza de los recursos de prueba.
- Se ha añadido el informe de resumen de ejecución de prueba.

IDT v1.1 para AWS IoT Greengrass v1.7.1

Notas de la versión:

- Se ha añadido compatibilidad para la integración de seguridad de hardware (HSI) de AWS IoT Greengrass.
- Se ha añadido compatibilidad para contenedor y sin contenedor de AWS IoT Greengrass.
- Se ha añadido la creación automatizada del rol de servicio de AWS IoT Greengrass.
- Se ha mejorado la limpieza de los recursos de prueba.
- Se ha añadido el informe de resumen de ejecución de prueba.

IDT v1.0 para AWS IoT Greengrass v1.6.1

Notas de la versión:

- Se ha añadido una corrección de errores de la prueba OTA para la compatibilidad con versiones de AWS IoT Greengrass futuras.

> **Note**
>
> Si utiliza IDT v1.0 para AWS IoT Greengrass v1.6.1, debe crear un rol de servicio de Greengrass. En versiones posteriores, IDT crea el rol de servicio.

Utilice IDT para ejecutar el conjunto de cualificación de AWS IoT Greengrass

Puede utilizar AWS IoT Device Tester (IDT) para AWS IoT Greengrass para verificar que el software de AWS IoT Greengrass Core se ejecuta en su hardware y puede comunicarse con la Nube de AWS. También realiza pruebas integrales con AWS IoT Core. Por ejemplo, verifica que su dispositivo pueda enviar y recibir mensajes MQTT y procesarlos correctamente.

Dado que AWS IoT Greengrass Version 1 ha pasado al modo de mantenimiento, IDT para AWS IoT Greengrass V1 ya no genera informes de cualificación firmados. Si desea añadir su hardware al catálogo de dispositivos AWS Partner, ejecute el conjunto de cualificación AWS IoT Greengrass V2 para generar informes de pruebas que puede enviar a AWS IoT. Para obtener más información, consulte AWSel Programa de cualificación de dispositivos y las versiones compatibles de IDT para AWS IoT Greengrass V2.

Además de los dispositivos de pruebas, IDT para AWS IoT Greengrass crea recursos (por ejemplo, elementos de AWS IoT, grupos de AWS IoT Greengrass, funciones de Lambda, etc.) en su Cuenta de AWS para facilitar el proceso de cualificación.

Para crear estos recursos, IDT para AWS IoT Greengrass utiliza las credenciales de AWS configuradas en el archivo `config.json` para realizar llamadas a la API en su nombre. Estos recursos se aprovisionarán en distintos momentos durante una prueba.

Al utilizar IDT para AWS IoT Greengrass para ejecutar el conjunto de cualificación de AWS IoT Greengrass, IDT lleva a cabo los siguientes pasos:

1. Carga y valida su dispositivo y las configuraciones de credenciales.

2. Realiza pruebas seleccionadas con los recursos locales y de la nube necesarios.
3. Depura los recursos locales y de la nube.
4. Genera informes de pruebas que indican si su dispositivo ha superado las pruebas necesarias para la cualificación.

Versiones del conjunto de pruebas

IDT para AWS IoT Greengrass organiza las pruebas en conjuntos de pruebas y grupos de pruebas.

- Un conjunto de pruebas es el conjunto de grupos de pruebas que se utiliza para verificar que un dispositivo funciona con versiones particulares de AWS IoT Greengrass.
- Un grupo de pruebas es el conjunto de pruebas individuales relacionadas con una característica concreta, como implementaciones de grupos de Greengrass y mensajería MQTT.

A partir de IDT v3.0.0, los conjuntos de pruebas incluyen control de versiones utilizando un formato *`major`*`.`*`minor`*`.`*`patch`*, por ejemplo `GGQ_1.0.0`. Al descargar IDT, el paquete incluye la versión más reciente del conjunto de pruebas.

> **Important**
>
> IDT admite las tres últimas versiones del conjunto de pruebas para la cualificación de dispositivos. Para obtener más información, consulte the section called “Política de compatibilidad de AWS IoT Device Tester para AWS IoT Greengrass V1”.
> Puede ejecutar `list-supported-products` para enumerar las versiones de AWS IoT Greengrass y conjuntos de pruebas compatibles con su versión actual de IDT. Las pruebas de versiones del conjunto de pruebas no compatibles no son válidas para la cualificación del dispositivo. IDT no imprime informes de cualificación para versiones no compatibles.

Actualizaciones de los parámetros de configuración de IDT

Las nuevas pruebas podrían introducir nuevas opciones de configuración de IDT.

- Si los ajustes son opcionales, IDT continúa ejecutando las pruebas.
- Si los ajustes son obligatorios, IDT se lo notifica y deja de ejecutarse. Después de configurar los ajustes, reinicie la ejecución de prueba.

Los ajustes de configuración se encuentran en la carpeta *<device-tester-extract-location>*`/configs`. Para obtener más información, consulte the section called “Configuración de los ajustes de IDT”.

Si una versión actualizada del conjunto de pruebas agrega ajustes de configuración, IDT crea una copia del archivo de configuración original en *<device-tester-extract-location>*`/configs`.

Descripciones de los grupos de pruebas

IDT v2.0.0 and later

Grupos de pruebas necesarias para la cualificación del núcleo

Estos grupos de pruebas son necesarios para cualificar su dispositivo AWS IoT Greengrass para el AWS Partner Device Catalog.

Dependencias de AWS IoT Greengrass Core

Valida que el dispositivo cumpla todos los requisitos de software y hardware del software AWS IoT Greengrass Core.

El caso de prueba `Software Packages Dependencies` de este grupo de pruebas no es aplicable cuando se realizan pruebas en un contenedor de Docker.

Implementación

Valida que las funciones de Lambda se puedan implementar en el dispositivo.

MQTT

Verifica la funcionalidad del enrutador de mensajes de AWS IoT Greengrass mediante la comprobación de la comunicación local entre el núcleo de Greengrass y los dispositivos cliente, que son dispositivos de IoT locales.

Over-the-Air (OTA)

Valida que el dispositivo pueda realizar correctamente una actualización OTA del software de un núcleo de AWS IoT Greengrass.

Este grupo de pruebas no es aplicable cuando se realizan pruebas en un contenedor de Docker.

Versión

Comprueba que la versión de AWS IoT Greengrass proporcionada sea compatible con la versión de AWS IoT Device Tester que está utilizando.

Grupos de pruebas opcionales

Estos grupos de pruebas son opcionales. Si opta por cualificar pruebas opcionales, el dispositivo se presenta con funciones adicionales en el AWS Partner Device Catalog.

Dependencias de contenedor

Valida que el dispositivo cumple todos los requisitos de software y hardware para ejecutar funciones de Lambda en modo contenedor en un núcleo de Greengrass.

Este grupo de pruebas no es aplicable cuando se realizan pruebas en un contenedor de Docker.

Contenedor de implementación

Valida que las funciones de Lambda se puedan implementar en el dispositivo y se ejecuten en modo contenedor en el núcleo de Greengrass.

Este grupo de pruebas no es aplicable cuando se realizan pruebas en un contenedor de Docker.

Dependencias de Docker (admitidas para IDT v2.2.0 y versiones posteriores)

Valida que el dispositivo cumple todas las dependencias técnicas necesarias para utilizar el conector de implementación de aplicaciones de Greengrass Docker para ejecutar contenedores.

Este grupo de pruebas no es aplicable cuando se realizan pruebas en un contenedor de Docker.

Integración de la seguridad por hardware (HSI)

Verifica que la biblioteca compartida de HSI proporcionada pueda interactuar con el módulo de seguridad de hardware (HSM) y se implementan correctamente las API de PKCS#11 necesarias. La biblioteca HSM y compartida debe poder firmar una CSR, realizar las operaciones de TLS y proporcionar las longitudes de clave y el algoritmo de clave pública correctos.

Dependencias de Stream Manager (admitidas para IDT v2.2.0 y versiones posteriores)

Valida que el dispositivo cumple todas las dependencias técnicas necesarias para ejecutar el administrador de transmisiones de AWS IoT Greengrass.

Dependencias de machine learning (compatible con IDT v3.1.0 y versiones posteriores)

Valida que el dispositivo cumpla todas las dependencias técnicas necesarias para realizar la inferencia de ML localmente.

Pruebas de inferencia de machine learning (compatibles con IDT v3.1.0 y versiones posteriores)

Valida que la inferencia de ML se pueda realizar en el dispositivo a prueba. Para obtener más información, consulte the section called “Opcional: Configuración del dispositivo para la cualificación ML”.

Pruebas de contenedores de inferencia de machine learning (compatible con IDT v3.1.0 y versiones posteriores)

Valida que la inferencia de ML se pueda realizar en el dispositivo a prueba y ejecutar en modo contenedor en un núcleo de Greengrass. Para obtener más información, consulte the section called “Opcional: Configuración del dispositivo para la cualificación ML”.

IDT v1.3.3 and earlier

Grupos de pruebas necesarias para la cualificación del núcleo

Estas pruebas son necesarias para cualificar el dispositivo AWS IoT Greengrass para el AWS Partner Device Catalog.

Dependencias de AWS IoT Greengrass Core

Valida que el dispositivo cumpla todos los requisitos de software y hardware del software AWS IoT Greengrass Core.

Combinación (interacción de seguridad de dispositivo)

Verifica la funcionalidad del administrador de certificados de dispositivo y la detección IP del dispositivo de núcleo de Greengrass cambiando la información de conectividad en el grupo de Greengrass en la nube. El grupo de pruebas rota el certificado de servidor de AWS IoT Greengrass y comprueba que AWS IoT Greengrass permita las conexiones.

Implementación (obligatoria para IDT v1.2 y versiones anteriores)

Valida que las funciones de Lambda se puedan implementar en el dispositivo.

Device Certificate Manager (DCM)

Verifica que el administrador de certificados del dispositivo AWS IoT Greengrass pueda generar un certificado de servidor durante el inicio y rotar certificados si están a punto de caducar.

Detección de IP (IPD)

Verifica que la información de conectividad del núcleo se actualice cuando hay cambios de dirección IP en un dispositivo de núcleo de Greengrass. Para obtener más información, consulte Activación de la detección automática de IP.

Registro

Verifica que el servicio de registro de AWS IoT Greengrass pueda escribir en un archivo de registro mediante una función de Lambda de usuario en Python.

MQTT

Verifica la funcionalidad del router de mensajes de AWS IoT Greengrass enviando mensajes sobre un tema que se dirige a dos funciones Lambda.

KCL

Verifica que AWS IoT Greengrass pueda ejecutar funciones Lambda nativas (compiladas).

Over-the-Air (OTA)

Este grupo de pruebas valida que su dispositivo pueda realizar correctamente una actualización OTA del software de AWS IoT Greengrass Core.

Penetración

Valida que el software AWS IoT Greengrass Core no se inicia si la protección de enlace físico/temporal y seccomp no están habilitadas. También se utilizar para verificar otras características relacionadas con la seguridad.

Sombra

Verifica la funcionalidad de sincronización de nube de sombra y sombra local.

Administrador de trabajos

Valida que los mensajes MQTT se pongan en cola con la configuración predeterminada del administrador de trabajos.

Servicio de intercambio de token (TES)

Verifica que AWS IoT Greengrass pueda intercambiar su certificado Core por credenciales válidas de AWS.

Versión

Comprueba que la versión de AWS IoT Greengrass proporcionada sea compatible con la versión de AWS IoT Device Tester que está utilizando.

Grupos de pruebas opcionales

Estas pruebas son opcionales. Si opta por cualificar pruebas opcionales, el dispositivo se presenta con funciones adicionales en el AWS Partner Device Catalog.

Dependencias de contenedor

Comprueba que el dispositivo cumpla todas las dependencias necesarias para ejecutar funciones de Lambda en el modo contenedor.

Integración de la seguridad por hardware (HSI)

Verifica que la biblioteca compartida de HSI proporcionada pueda interactuar con el módulo de seguridad de hardware (HSM) y se implementan correctamente las API de PKCS#11 necesarias. La biblioteca HSM y compartida debe poder firmar una CSR, realizar las operaciones de TLS y proporcionar las longitudes de clave y el algoritmo de clave pública correctos.

Acceso a recursos locales

Verifica la característica de acceso al recurso local (LRA) de AWS IoT Greengrass proporcionando acceso a archivos y directorios locales propiedad de diversos usuarios y grupos de Linux a funciones de Lambda en contenedores a través de las API LRA de AWS IoT Greengrass. Las funciones de Lambda deben permitir o denegar el acceso a los recursos locales en función de la configuración del acceso al recurso local.

Red

Verifica que se puedan establecer conexiones de conector desde una función de Lambda. Estas conexiones de socket se deben permitir o denegar en función de la configuración del núcleo de Greengrass.

Requisitos previos para ejecutar el paquete de AWS IoT Greengrass calificación

En esta sección se describen los requisitos previos para utilizar AWS IoT Device Tester (IDT) AWS IoT Greengrass para ejecutar el conjunto de requisitos. AWS IoT Greengrass

Descargue la última versión de Device Tester para AWS IoTAWS IoT Greengrass

Descargue la última versión de IDT y extraiga el software en una ubicación de su sistema de archivos en la que tenga permisos de lectura y escritura.

> **Note**
>
> IDT no admite la ejecución por parte de varios usuarios desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Le recomendamos que extraiga el paquete IDT en una unidad local y ejecute el binario IDT en su estación de trabajo local.
> Windows tiene una limitación de longitud de ruta de 260 caracteres. Si utiliza Windows, extraiga IDT en un directorio raíz como `C:\` o `D:\` para mantener las rutas por debajo del límite de 260 caracteres.

Cree y configure un Cuenta de AWS

Antes de poder utilizar IDT para AWS IoT Greengrass, debe realizar los siguientes pasos:

1. Cree un Cuenta de AWS. Si ya tiene uno Cuenta de AWS, vaya al paso 2.
2. Configurar permisos de IDT.

Estos permisos de cuenta permiten a IDT acceder a AWS los servicios y crear AWS recursos, como AWS IoT cosas, grupos de Greengrass y funciones de Lambda, en su nombre.

Para crear estos recursos, IDT for AWS IoT Greengrass utiliza AWS las credenciales configuradas en el `config.json` archivo para realizar llamadas a la API en su nombre. Estos recursos se aprovisionarán en distintos momentos durante una prueba.

> **Note**
>
> Aunque la mayoría de las pruebas cumplen los requisitos para el Nivel gratuito de Amazon Web Services, debe proporcionar una tarjeta de crédito cuando se cree una Cuenta de AWS. Para obtener más información, consulte ¿Por qué necesito un método de pago si mi cuenta está cubierta por la capa gratuita?.

Paso 1: Crea una Cuenta de AWS

En este paso, cree y configure una Cuenta de AWS. Si ya tiene uno Cuenta de AWS, vaya athe section called “Paso 2: Configurar los permisos de IDT”.

Inscríbase en una Cuenta de AWS

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

Para suscribirte a una Cuenta de AWS

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

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

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

AWS te envía un correo electrónico de confirmación una vez finalizado el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en https://aws.amazon.com/ y seleccionando Mi cuenta.

Creación de un usuario con acceso administrativo

Después de crear un usuario administrativo Cuenta de AWS, asegúrelo Usuario raíz de la cuenta de AWS AWS IAM Identity Center, habilite y cree un usuario administrativo para no usar el usuario root en las tareas diarias.

Proteja su Usuario raíz de la cuenta de AWS

1. Inicie sesión AWS Management Consolecomo propietario de la cuenta seleccionando el usuario root e introduciendo su dirección de Cuenta de AWS correo electrónico. En la siguiente página, escriba su contraseña.

   Para obtener ayuda para iniciar sesión con el usuario raíz, consulte Signing in as the root user en la Guía del usuario de AWS Sign-In .
2. Active la autenticación multifactor (MFA) para el usuario raíz.

   Para obtener instrucciones, consulte Habilitar un dispositivo MFA virtual para el usuario Cuenta de AWS raíz (consola) en la Guía del usuario de IAM.

Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

   Consulte las instrucciones en Activar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center .
2. En IAM Identity Center, conceda acceso administrativo a un usuario.

   Para ver un tutorial sobre su uso Directorio de IAM Identity Center como fuente de identidad, consulte Configurar el acceso de los usuarios con la configuración predeterminada Directorio de IAM Identity Center en la Guía del AWS IAM Identity Center usuario.

Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

  Para obtener ayuda para iniciar sesión con un usuario del Centro de identidades de IAM, consulte Iniciar sesión en el portal de AWS acceso en la Guía del AWS Sign-In usuario.

Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

Para conocer las instrucciones, consulte Create a permission set en la Guía del usuario de AWS IAM Identity Center .

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

   Para conocer las instrucciones, consulte Add groups en la Guía del usuario de AWS IAM Identity Center .

Paso 2: Configurar los permisos de IDT

En este paso, configure los permisos que IDT for AWS IoT Greengrass utiliza para ejecutar pruebas y recopilar datos de uso de IDT. Puede usar AWS Management Console o AWS Command Line Interface (AWS CLI) para crear una política de IAM y un usuario de prueba para IDT y, a continuación, adjuntar políticas al usuario. Si ya ha creado un usuario de prueba para IDT, vaya a the section called “Configure su dispositivo para ejecutar pruebas de IDT” o the section called “Opcional: Configuración del contenedor Docker”.

- Configuración de permisos para IDT (consola)
- Configuración de permisos para IDT (AWS CLI)

Configuración de permisos de IDT (consola)

Siga estos pasos para usar la consola para configurar permisos para IDT para AWS IoT Greengrass.

1. Inicie sesión en la consola de IAM.
2. Crear una política administrada que conceda permisos para crear roles con permisos específicos.

   a. En el panel de navegación, seleccione Políticas y, a continuación, Crear política.

   b. En la pestaña JSON, reemplace el contenido del marcador de posición por la política siguiente.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ManageRolePoliciesForIDTGreengrass",
            "Effect": "Allow",
```

```
            "Action": [
                "iam:DetachRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ],
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
                        "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
                        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                    ]
                }
            }
        },
        {
            "Sid": "ManageRolesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:PassRole",
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ]
        }
    ]
}
```

Important

La siguiente política concede permiso para crear y administrar los roles precisados por IDT para AWS IoT Greengrass. Esto incluye permisos para adjuntar las siguientes políticas AWS administradas:

- AWSGreengrassResourceAccessRolePolicy
- Grassota verde UpdateArtifactAccess
- AWSLambdaBasicExecutionRole

c. Elija Siguiente: Etiquetas.

d. Elija Siguiente: Revisar.

e. En Nombre, escriba **`IDTGreengrassIAMPermissions`**. En Summary (Resumen), revise los permisos concedidos por la política.

f. Elija Crear política.

3. Cree un usuario de IAM y adjunte los permisos requeridos por IDT para AWS IoT Greengrass.

a. Cree un usuario de IAM. Siga los pasos del 1 al 5 en Creación de usuarios de IAM (consola) en la Guía del usuario de IAM.

b. Adjunte los permisos a su usuario de IAM:

i. En la página Establecer permisos, elija Adjuntar políticas existentes directamente.

ii. Busque la política IDTGreengrassIAMPermissions que ha creado en el paso anterior. Seleccione la casilla de verificación.

iii. Busca la política. AWSIoTDeviceTesterForGreengrassFullAccess Seleccione la casilla de verificación.

> **Note**
>
> AWSIoTDeviceTesterForGreengrassFullAccessSe trata de una política AWS gestionada que define los permisos que IDT necesita para crear y acceder a AWS los recursos utilizados para las pruebas. Para obtener más información, consulte the section called “AWS política gestionada para IDT”.

c. Elija Next: Tags (Siguiente: Etiquetas).

d. Elija Next: Review (Siguiente: revisar) para ver un resumen de sus opciones.

e. Seleccione la opción Crear un usuario.

f. Para ver las claves de acceso del usuario (ID de clave de acceso y claves de acceso secretas), elija Show (Mostrar) junto a la contraseña y la clave de acceso. Para guardar las claves de acceso, elija Download.csv (Descargar archivo .csv) y, a continuación, guarde el

archivo en un lugar seguro. Utilice esta información más adelante para configurar su archivo de credenciales de AWS .

4. Siguiente paso: Configure su dispositivo físico.

Configuración de permisos de IDT (AWS CLI)

Siga estos pasos para configurar AWS CLI los permisos de IDT. AWS IoT Greengrass Si ya ha configurado permisos en la consola, vaya a the section called “Configure su dispositivo para ejecutar pruebas de IDT” o the section called “Opcional: Configuración del contenedor Docker”.

1. En su ordenador, instale y configure el AWS CLI si aún no está instalado. Siga los pasos que se indican en Instalación de la AWS CLI en la Guía del usuario de la AWS Command Line Interface .

   > **Note**
   >
   > AWS CLI Se trata de una herramienta de código abierto que puede utilizar para interactuar con los AWS servicios desde el shell de la línea de comandos.

2. Cree una política administrada por el cliente que conceda permisos para administrar IDT y roles de AWS IoT Greengrass .

   Linux, macOS, or Unix

   ```
   aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
   document '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "ManageRolePoliciesForIDTGreengrass",
               "Effect": "Allow",
               "Action": [
                   "iam:DetachRolePolicy",
                   "iam:AttachRolePolicy"
               ],
               "Resource": [
                   "arn:aws:iam::*:role/idt-*",
                   "arn:aws:iam::*:role/GreengrassServiceRole"
   ```

```
            ],
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
                        "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
                        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                    ]
                }
            }
        },
        {
            "Sid": "ManageRolesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:PassRole",
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ]
        }
    ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --
policy-document '{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid
\": \"ManageRolePoliciesForIDTGreengrass\",\"Effect\": \"Allow\",
\"Action\": [\"iam:DetachRolePolicy\", \"iam:AttachRolePolicy\"],
 \"Resource\": [\"arn:aws:iam::*:role/idt-*\",\"arn:aws:iam::*:role/
GreengrassServiceRole\"],\"Condition\": {\"ArnEquals\": {\"iam:PolicyARN\":
 [\"arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
\",\"arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess
\",\"arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole\"]}}},
{\"Sid\": \"ManageRolesForIDTGreengrass\",\"Effect\": \"Allow\",\"Action\":
```

```
 [\"iam:CreateRole\",\"iam:DeleteRole\", \"iam:PassRole\", \"iam:GetRole
\"],\"Resource\": [\"arn:aws:iam::*:role/idt-*\",\"arn:aws:iam::*:role/
GreengrassServiceRole\"]}]}'
```

> **Note**
>
> Este paso incluye un ejemplo de símbolo del sistema de Windows porque utiliza una sintaxis JSON diferente a la de los comandos de terminal Linux, macOS o Unix.

3. Cree un usuario de IAM y adjunte los permisos requeridos por IDT para AWS IoT Greengrass.

   a. Cree un usuario de IAM. En esta configuración de ejemplo, el usuario se denomina `IDTGreengrassUser`.

   ```
   aws iam create-user --user-name IDTGreengrassUser
   ```

   b. Asocie la política `IDTGreengrassIAMPermissions` que creó en el paso 2 a su usuario de IAM. Sustituya <account-id>el comando por el ID de su. Cuenta de AWS

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

   c. Asocie la política `AWSIoTDeviceTesterForGreengrassFullAccess` a su usuario de IAM.

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::aws:policy/AWSIoTDeviceTesterForGreengrassFullAccess
   ```

   > **Note**
   >
   > AWSIoTDeviceTesterForGreengrassFullAccessSe trata de una política AWS gestionada que define los permisos que IDT necesita para crear y acceder a AWS los recursos utilizados para las pruebas. Para obtener más información, consulte the section called “AWS política gestionada para IDT”.

4. Cree una clave de acceso secreta para el usuario.

   ```
   aws iam create-access-key --user-name IDTGreengrassUser
   ```

Almacene la salida en una ubicación segura. Esta información se utiliza más adelante para configurar el archivo de AWS credenciales.

5. Siguiente paso: Configure su dispositivo físico.

AWS política gestionada para AWS IoT Device Tester

La política AWSIoTDeviceTesterForGreengrassFullAccessgestionada permite a IDT ejecutar operaciones y recopilar métricas de uso. Esta política concede los siguientes permisos de IDT:

- `iot-device-tester:CheckVersion`. Compruebe si un conjunto de versiones AWS IoT Greengrass, el conjunto de pruebas y las versiones de IDT son compatibles.
- `iot-device-tester:DownloadTestSuite`. Descargue los conjuntos de pruebas.
- `iot-device-tester:LatestIdt`. Obtenga información sobre la última versión de IDT que está disponible para descarga.
- `iot-device-tester:SendMetrics`. Publique los datos de uso que IDT recopila sobre las pruebas.
- `iot-device-tester:SupportedVersion`. Obtenga la lista de AWS IoT Greengrass las versiones de los conjuntos de pruebas compatibles con IDT. Esta información se muestra en la ventana de línea de comandos.

Configure su dispositivo para ejecutar pruebas de IDT

Para configurar el dispositivo debe instalar dependencias de AWS IoT Greengrass, configurar el software de AWS IoT Greengrass Core, configurar su equipo host para acceder a su dispositivo y configurar los permisos de usuario en su dispositivo.

Verificación de las dependencias de AWS IoT Greengrass en el dispositivo sometido a prueba

Antes de que IDT para AWS IoT Greengrass pueda probar los dispositivos, asegúrese de que el dispositivo esté configurado tal como se describe en Introducción a AWS IoT Greengrass. Para obtener información sobre las plataformas admitidas, consulte Plataformas compatibles.

Configuración del software de AWS IoT Greengrass

IDT para AWS IoT Greengrass prueba su dispositivo para comprobar la compatibilidad con una versión específica de AWS IoT Greengrass. IDT le ofrece dos opciones para probar AWS IoT Greengrass en sus dispositivos:

- Descargue y utilice una versión del software de AWS IoT Greengrass Core. IDT instala el software.
- Utilice una versión del software de AWS IoT Greengrass Core ya instalada en su dispositivo.

> **Note**
>
> Cada versión de AWS IoT Greengrass tiene una versión correspondiente de IDT. Debe descargar la versión de IDT que corresponde a la versión de AWS IoT Greengrass que está utilizando.

Estas opciones se describen en las siguientes secciones. Solo tiene que hacer una.

Opción 1: Descargar el software AWS IoT Greengrass Core y configurar AWS IoT Device Tester para que lo utilice

Puede descargar el software AWS IoT Greengrass Core desde la página de descargas del Software AWS IoT Greengrass Core..

1. Busque la arquitectura y la distribución de Linux correctas y, a continuación, elija Download (Descargar).
2. Copie el archivo tar.gz en *<device-tester-extract-location>*`/products/greengrass/ggc`.

> **Note**
>
> No cambie el nombre del archivo tar.gz de AWS IoT Greengrass. No coloque varios archivos en este directorio para el mismo sistema operativo y arquitectura. Por ejemplo, tener los archivos `greengrass-linux-armv7l-1.7.1.tar.gz` y `greengrass-linux-armv7l-1.8.1.tar.gz` en dicho directorio hará que las pruebas devuelvan un error.

Opción 2: Utilizar una instalación existente de AWS IoT Greengrass con AWS IoT Device Tester

Configure IDT para probar el software de AWS IoT Greengrass Core instalado en su dispositivo añadiendo el atributo `greengrassLocation` al archivo `device.json` ubicado en la carpeta *`<device-tester-extract-location>`*`/configs`. Por ejemplo:

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

Para obtener más información acerca del archivo `device.json`, consulte Configurar device.json.

En los dispositivos Linux, la ubicación predeterminada del software Núcleo de AWS IoT Greengrass es `/greengrass`.

> **Note**
>
> Su dispositivo debe tener una instalación del software de AWS IoT Greengrass Core que no se haya iniciado.
> Asegúrese de haber añadido el usuario de `ggc_user` y `ggc_group` en el dispositivo. Para obtener más información, consulte Configuración del entorno para AWS IoT Greengrass.

Configuración del equipo host para acceder a un dispositivo en pruebas

IDT se ejecuta en su equipo host y debe poder utilizar SSH para conectarse a su dispositivo. Existen dos opciones para permitir que IDT obtenga acceso SSH a los dispositivos sometidos a la prueba:

1. Siga las instrucciones que se indican aquí para crear un par de claves SSH y autorizar su clave para iniciar sesión en su dispositivo en proceso de prueba sin especificar una contraseña.
2. Proporcione un nombre de usuario y una contraseña para cada dispositivo en el archivo `device.json`. Para obtener más información, consulte Configurar device.json.

Puede utilizar cualquier implementación SSL para crear una clave SSH. Las siguientes instrucciones muestran cómo utilizar SSH-KEYGEN o PuTTYgen (para Windows). Si utiliza otra implementación de SSL, consulte la documentación de dicha aplicación.

IDT utiliza claves SSH para autenticar con su dispositivo bajo prueba.

Para crear una clave SSH con SSH-KEYGEN, realice el siguiente procedimiento:

1. Cree una clave de SSH.

   Puede utilizar el comando ssh-keygen de Open SSH para crear un par de claves SSH. Si ya tiene un par de claves SSH en su equipo host, es una práctica recomendada crear un par de claves SSH específicamente para IDT. De esta forma, una vez completadas las pruebas, el equipo host ya no podrá conectarse a su dispositivo sin introducir una contraseña. También le permite restringir el acceso al dispositivo remoto únicamente a aquellos que lo necesiten.

   > **Note**
   >
   > Windows no tiene instalado un cliente SSH. Para obtener más información sobre la instalación de un cliente SSH en Windows, consulte Download SSH Client Software.

   El comando ssh-keygen le solicita un nombre y la ruta para almacenar el par de claves. De forma predeterminada, los archivos de par de claves se denominan `id_rsa` (clave privada) y `id_rsa.pub` (clave pública). En macOS y Linux, la ubicación predeterminada de estos archivos es `~/.ssh/`. En Windows, la ubicación predeterminada es `C:\Users\`*`<user-name>`*`\.ssh`.

   Cuando se le solicite, introduzca una frase clave para proteger la clave SSH. Para obtener más información, consulte la sección acerca de cómo generar una nueva clave SSH.

2. Añada claves SSH autorizadas a su dispositivo en proceso de prueba.

   IDT debe utilizar su clave privada de SSH para iniciar sesión en el dispositivo a prueba. Para autorizar que su clave privada de SSH inicie sesión en el dispositivo a prueba, use el comando ssh-copy-id en su equipo host. Este comando añade su clave pública al archivo `~/.ssh/authorized_keys` que se encuentra en su dispositivo a prueba. Por ejemplo:

   $ ssh-copy-id ***<remote-ssh-user>***@***<remote-device-ip>***

   Donde *`usuario-ssh-remoto`* es el nombre de usuario utilizado para iniciar sesión en su dispositivo sometido a ensayo e *`ip-dispositivo-remoto`* es la dirección IP del dispositivo sometido a ensayo para ejecutar pruebas. Por ejemplo:

   ssh-copy-id pi@192.168.1.5

Cuando se le solicite, introduzca la contraseña para el nombre de usuario que ha especificado en el comando ssh-copy-id.

ssh-copy-id presupone que la clave pública se denomina `id_rsa.pub` y se almacena en la ubicación predeterminada (en macOS y Linux, `~/.ssh/` y en Windows, `C:\Users\`*`<user-name>`*`\.ssh`). Si asignó a la clave pública un nombre diferente o la almacenó en otra ubicación, debe especificar la ruta completa a su clave pública SSH utilizando la opción -i para ssh-copy-id (por ejemplo: ssh-copy-id -i ~/my/path/myKey.pub). Para obtener más información acerca de la creación de claves de SSH y la copia de las claves públicas, consulte SSH-COPY-ID.

Para crear una clave SSH con PuTTYgen (solo Windows), realice el siguiente procedimiento:

1. Asegúrese de que tiene el servidor y el cliente de OpenSSH instalados en su dispositivo en proceso de prueba. Para obtener más información, consulte OpenSSH.
2. Instale PuTTYgen en su dispositivo en proceso de prueba.
3. Abra PuTTYgen.
4. Elija Generate (Generar) y mueva el cursor del ratón dentro del cuadro para generar una clave privada.
5. En el menú Conversions (Conversiones), elija Export OpenSSH key (Exportar clave OpenSSH) y guarde la clave privada con una extensión de archivo `.pem`.
6. Añada la clave pública al archivo `/home/`*`<user>`*`/.ssh/authorized_keys` en el dispositivo en proceso de prueba.

   a. Copie el texto de la clave pública de la ventana PuTTYgen.

   b. Utilice PuTTY para crear una sesión en su dispositivo en proceso de prueba.

      i. En un símbolo del sistema o en una ventana de Windows PowerShell, ejecute el siguiente comando:

         C:/***<path-to-putty>***/putty.exe -ssh ***<user>***@***<dut-ip-address>***

      ii. Cuando se le solicite, escriba la contraseña de su dispositivo.

      iii. Utilice vi u otro editor de texto para añadir la clave pública al archivo `/home/`*`<user>`*`/.ssh/authorized_keys` en su dispositivo en proceso de prueba.
7. Actualice el archivo `device.json` con su nombre de usuario, la dirección IP y la ruta al archivo de clave privada que acaba de guardar en el equipo host para cada dispositivo en proceso de

prueba. Para obtener más información, consulte the section called “Configurar device.json”. Asegúrese de proporcionar la ruta completa y el nombre de archivo a la clave privada y utilizar barras diagonales ("/"). Por ejemplo, para la ruta de Windows `C:\DT\privatekey.pem`, utilice `C:/DT/privatekey.pem` en el archivo `device.json`.

Configuración de los permisos de usuario en el dispositivo

IDT realiza operaciones en diversos directorios y archivos en un dispositivo que se está probando. Algunas de estas operaciones requieren permisos elevados (usando sudo). Para automatizar estas operaciones, IDT para AWS IoT Greengrass debe ser capaz de ejecutar comandos con sudo sin que se le solicite una contraseña.

Siga estos pasos en el dispositivo a prueba para permitir acceso a sudo sin que se le solicite una contraseña.

> **Note**
>
> `username` hace referencia al usuario de SSH que utiliza IDT para obtener acceso al dispositivo bajo prueba.

Para añadir el usuario al grupo sudo

1. En el dispositivo bajo prueba, ejecute `sudo usermod -aG sudo` *`<username>`*.
2. Cierre la sesión y, a continuación, vuelva a iniciar sesión para que los cambios surtan efecto.
3. Para comprobar que su nombre de usuario se haya añadido correctamente, ejecute sudo echo test. Si no se le solicita una contraseña, el usuario se ha configurado correctamente.
4. Añada el archivo `/etc/sudoers` y agregue la siguiente línea al final del archivo:

   `<ssh-username>` `ALL=(ALL) NOPASSWD: ALL`

Configurar el dispositivo para probar características opcionales

En los temas siguientes se describe cómo configurar los dispositivos para ejecutar pruebas de IDT para características opcionales. Siga estos pasos de configuración solo si desea probar estas características. De lo contrario, siga en the section called “Configuración de los ajustes de IDT”.

Temas

Opcional: Configuración del contenedor Docker para IDT para AWS IoT Greengrass

AWS IoT Greengrass proporciona un Dockerfile y una imagen de Docker que facilitan la ejecución del software de AWS IoT Greengrass Core en un contenedor Docker. Después de configurar el contenedor de AWS IoT Greengrass, puede ejecutar pruebas IDT. Actualmente, solo se admiten arquitecturas Docker x86_64 para ejecutar IDT AWS IoT Greengrass.

Esta función requiere IDT v2.3.0 o posterior.

El proceso de configuración del contenedor Docker para ejecutar pruebas IDT depende de si se utiliza la imagen Docker o Dockerfile proporcionada por AWS IoT Greengrass.

- Uso de la imagen de Docker. La imagen de Docker con el software de AWS IoT Greengrass Core y las dependencias instalados.
- Utilice el archivo Dockerfile. El archivo Dockerfile contiene código fuente que puede usar para crear imágenes de contenedor de AWS IoT Greengrass personalizadas. La imagen se puede modificar para ejecutarla en arquitecturas de plataforma distintas o para reducir su tamaño.

> **Note**
>
> AWS IoT Greengrass no proporciona un Dockerfile o una imagen de Docker para la versión 1.11.1 del software principal AWS IoT Greengrass. Para ejecutar pruebas IDT en sus propias imágenes de contenedor personalizadas, la imagen debe incluir las dependencias definidas en el archivo Dockerfile proporcionado por AWS IoT Greengrass.

Las siguientes características no están disponibles cuando se ejecuta AWS IoT Greengrass en un contenedor Docker:

- Conectores que se ejecutan en el modo Contenedor de Greengrass. Para ejecutar un conector en un contenedor de Docker, el conector debe ejecutarse en modo Sin contenedor. Para buscar conectores compatibles con el modo Sin contenedor, consulte the section called “conectores de Greengrass proporcionados por AWS”. Algunos de estos conectores tienen un parámetro de modo de aislamiento que debe establecer en Sin contenedor.

- Dispositivos locales y recursos de volumen. Las funciones de Lambda definidas por el usuario que se ejecutan en el contenedor de Docker deben obtener acceso directamente a los dispositivos y volúmenes del dispositivo principal.

Configuración de la imagen de Docker proporcionada por AWS IoT Greengrass

Siga estos pasos para configurar la imagen de Docker de AWS IoT Greengrass para ejecutar pruebas IDT.

Requisitos previos

Antes de empezar este tutorial, debe hacer lo siguiente.

- Debe instalar el software y las versiones siguientes en su ordenador host en función de la versión de la AWS Command Line Interface (AWS CLI) que elija.

  AWS CLI version 2

  - Docker, versión 18.09 o superior. Es posible que las versiones anteriores también funcionen, pero recomendamos la 18.09 o una versión posterior.
  - AWS CLI versión 2.0.0 o posterior
    - Para instalar la versión 2 de la AWS CLI, consulte Instalación de la AWS CLI versión 2.
    - Para configurar el AWS CLI, consulte Configuración de la AWS CLI.

    > **Note**
    >
    > Para actualizar a una versión 2 de AWS CLI posterior en un equipo con Windows, debe repetir el proceso de instalación de MSI.

  AWS CLI version 1

  - Docker, versión 18.09 o superior. Es posible que las versiones anteriores también funcionen, pero recomendamos la 18.09 o una versión posterior.
  - Python, versión 3.6 o superior.
  - pip, versión 18.1 o posterior.
  - AWS CLI versión 1.17.10 o posterior
    - Para instalar la versión 1 de AWS CLI, consulte Instalación de la AWS CLI versión 1.
    - Para configurar el AWS CLI, consulte Configuración de la AWS CLI.

- Para actualizar a la versión más reciente de la versión 1 de AWS CLI, ejecute el siguiente comando.

```
pip install awscli --upgrade --user
```

> **Note**
>
> Si utiliza la instalación de MSI de la AWS CLI versión 1 en Windows, tenga en cuenta lo siguiente:
>
> - Si la instalación de la versión 1 de AWS CLI no consigue instalar botocore, intente utilizar la instalación de Python y pip.
> - Para actualizar a una versión 1 de AWS CLI posterior, deberá repetir el proceso de instalación del MSI.

- Para acceder a los recursos de Amazon Elastic Container Registry (Amazon ECR), debe conceder el siguiente permiso.
  - Amazon ECR requiere que los usuarios tengan permiso `ecr:GetAuthorizationToken` para realizar llamadas a la API AWS Identity and Access Management a través de una política de IAM antes de que puedan autenticarse en un registro, así como insertar o extraer imágenes de cualquier repositorio de Amazon ECR. Para obtener más información, consulte los ejemplos de políticas de repositorios de Amazon ECR y el Acceso a un repositorio de Amazon ECR en la Guía del usuario de Amazon Elastic Container Registry.

1. Descargar la imagen de Docker y configurar el contenedor. Puede descargar la imagen preinstalada de Docker Hub o Amazon Elastic Container Registry(Amazon ECR) y ejecutarla en plataformas de Windows, macOS y Linux (x86_64).

   Para descargar la imagen de Docker de Amazon ECR, complete todos los pasos de the section called “Obtener la imagen del contenedor AWS IoT Greengrass de Amazon ECR”. A continuación, vuelva a este tema para continuar con la configuración.
2. Solo usuarios de Linux: asegúrese de que el usuario que ejecuta IDT tiene permiso para ejecutar comandos Docker. Para obtener más información, consulte Manage Docker as a non-root user en la documentación de Docker.

3. Para ejecutar el contenedor de AWS IoT Greengrass, utilice el comando para su sistema operativo:

   Linux

   ```
   docker run --rm --init -it -d --name aws-iot-greengrass \
   -p 8883:8883 \
   -v <host-path-to-kernel-config-file>:<container-path> \
   <image-repository>:<tag>
   ```

   - Sustituya *<host-path-to-kernel-config-file>* por la ruta al archivo de configuración del kernel en el host y *<container-path>* por la ruta en la que el volumen se monta en el contenedor.

     El archivo de configuración del kernel en el host generalmente se encuentra en `/proc/config.gz` o `/boot/config-`*<kernel-release-date>*. Puede ejecutar `uname -r` para encontrar el valor *<kernel-release-date>*.

     Ejemplo: para montar el archivo de configuración desde `/boot/config-`*<kernel-release-date>*

     ```
     -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
     ```

     Ejemplo: para montar el archivo de configuración desde `proc/config.gz`

     ```
     -v /proc/config.gz:/proc/config.gz \
     ```

   - Sustituya *<image-repository>*:*<tag>* en el comando con el nombre del repositorio y la etiqueta de la imagen objetivo.

     Ejemplo: para señalar la versión más reciente del software AWS IoT Greengrass Core

     ```
     216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
     ```

     Para obtener la lista de imágenes de Docker de AWS IoT Greengrass, ejecute el siguiente comando.

     ```
     aws ecr list-images --region us-west-2 --registry-id 216483018798 --
     repository-name aws-iot-greengrass
     ```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- Sustituya *<image-repository>*:*<tag>* en el comando con el nombre del repositorio y la etiqueta de la imagen objetivo.

  Ejemplo: para señalar la versión más reciente del software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obtener la lista de imágenes de Docker de AWS IoT Greengrass, ejecute el siguiente comando:

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- Sustituya *<image-repository>*:*<tag>* en el comando con el nombre del repositorio y la etiqueta de la imagen objetivo.

  Ejemplo: para señalar la versión más reciente del software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obtener la lista de imágenes de Docker de AWS IoT Greengrass, ejecute el siguiente comando:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

> **Important**
>
> Al hacer pruebas con IDT, no incluya el argumento `--entrypoint /greengrass-entrypoint.sh \` que se utiliza para ejecutar la imagen para el uso AWS IoT Greengrass en general.

4. Siguiente paso: configure sus credenciales de AWS y el archivo `device.json`.

Configuración del archivo Dockerfile proporcionado por AWS IoT Greengrass

Siga estos pasos para configurar la imagen Docker creada desde Dockerfile de AWS IoT Greengrass para ejecutar pruebas IDT.

1. Desde the section called “AWS IoT Greengrass Software Docker”, descargue el paquete Dockerfile en su equipo host y extráigalo.
2. Abrir `README.md`. Los tres pasos siguientes hacen referencia a las secciones de este archivo.
3. Asegúrese de que cumpla los requisitos de la sección Requisitos previos.
4. Solo usuarios de Linux: complete los pasos Habilitar protección de enlaces permanentes y simbólicos y Habilitar reenvío de red IPv4.
5. Para crear la imagen de Docker, complete todos los pasos del Paso 1. Cree la imagen de AWS IoT Greengrass Docker. A continuación, vuelva a este tema para continuar con la configuración.
6. Para ejecutar el contenedor de AWS IoT Greengrass, utilice el comando para su sistema operativo:

   Linux

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
-v <host-path-to-kernel-config-file>:<container-path> \
<image-repository>:<tag>
```

- Sustituya *<host-path-to-kernel-config-file>* por la ruta al archivo de configuración del kernel en el host y *<container-path>* por la ruta en la que el volumen se monta en el contenedor.

  El archivo de configuración del kernel en el host generalmente se encuentra en `/proc/config.gz` o `/boot/config-`*<kernel-release-date>*. Puede ejecutar `uname -r` para encontrar el valor *<kernel-release-date>*.

  Ejemplo: para montar el archivo de configuración desde `/boot/config-`*<kernel-release-date>*

  ```
  -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
  ```

  Ejemplo: para montar el archivo de configuración desde `proc/config.gz`

  ```
  -v /proc/config.gz:/proc/config.gz \
  ```

- Sustituya *<image-repository>*:*<tag>* en el comando con el nombre del repositorio y la etiqueta de la imagen objetivo.

  Ejemplo: para señalar la versión más reciente del software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obtener la lista de imágenes de Docker de AWS IoT Greengrass, ejecute el siguiente comando.

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- Sustituya *<image-repository>*:*<tag>* en el comando con el nombre del repositorio y la etiqueta de la imagen objetivo.

  Ejemplo: para señalar la versión más reciente del software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obtener la lista de imágenes de Docker de AWS IoT Greengrass, ejecute el siguiente comando:

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- Sustituya *<image-repository>*:*<tag>* en el comando con el nombre del repositorio y la etiqueta de la imagen objetivo.

  Ejemplo: para señalar la versión más reciente del software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obtener la lista de imágenes de Docker de AWS IoT Greengrass, ejecute el siguiente comando:

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

> **Important**
>
> Al hacer pruebas con IDT, no incluya el argumento `--entrypoint /greengrass-entrypoint.sh \` que se utiliza para ejecutar la imagen para el uso AWS IoT Greengrass en general.

7. Siguiente paso: configure sus credenciales de AWS y el archivo `device.json`.

Solución de problemas con la configuración del contenedor Docker en IDT para AWS IoT Greengrass

Utilice la información siguiente como ayuda para solucionar problemas relacionados con la ejecución de un contenedor de Docker para IDT para las pruebas de AWS IoT Greengrass.

ADVERTENCIA: Error al cargar el archivo de configuración:/home/user/.docker/config.json - stat / home/<user>/.docker/config.json: permiso denegado

Si obtiene este error al ejecutar comandos de `docker` en Linux, ejecute el siguiente comando. Sustituya*`<user>`* en el siguiente comando con el usuario que ejecuta IDT.

```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rwx /home/<user>/.docker -R
```

Opcional: Configuración del dispositivo para la cualificación ML

IDT para AWS IoT Greengrass proporciona pruebas de cualificación de machine learning (ML) para validar que sus dispositivos pueden realizar inferencia de ML localmente utilizando modelos entrenados en la nube.

Para ejecutar pruebas de cualificación de ML, primero debe configurar los dispositivos como se describe en the section called “Configure su dispositivo para ejecutar pruebas de IDT”. A continuación, siga los pasos de este tema para instalar dependencias para los marcos de ML que desea ejecutar.

Es necesario IDT v3.1.0 o posterior para realizar pruebas de cualificación de ML.

Instalación de dependencias del marco de ML

Todas las dependencias del marco de ML deben instalarse en el directorio `/usr/local/lib/python3.x/site-packages`. Para asegurarse de que están instalados en el directorio correcto, le

recomendamos que utilice permisos raíz de `sudo` al instalar las dependencias. Los entornos virtuales no son compatibles con las pruebas de cualificación.

> **Note**
>
> Si está probando funciones de Lambda que se ejecutan con la creación de contenedores (en modo contenedor Greengrass), la creación de enlaces simbólicos para las bibliotecas Python en `/usr/local/lib/python3.x` no se admite. Para evitar errores, debe instalar las dependencias en el directorio correcto.

Siga los pasos para instalar las dependencias para su marco de destino:

- Instalar dependencias de MXNet
- the section called “Instalar dependencias de TensorFlow”
- Instalar dependencias DLR

Instalar dependencias de Apache MXNet

Las pruebas de cualificación IDT para este marco tienen las siguientes dependencias:

- Python 3.6 o Python 3.7.

  > **Note**
  >
  > Si está utilizando Python 3.6, debe crear un enlace simbólico desde binarios de Python 3.7 a Python 3.6. Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass. Por ejemplo:
  >
  > ```
  > sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
  > ```

- Apache MXNet v1.2.1 o posterior.
- NumPy. La versión debe ser compatible con su versión MXNet.

Instalación de MXNet

Siga las instrucciones de la documentación de MXNet para instalar MXNet.

> **Note**
>
> Si en el dispositivo están instalados Python 2.x y Python 3.x, use Python 3.x en los comandos que ejecute para instalar las dependencias.

Validación de la instalación de MXNet

Elija una de las siguientes opciones para validar la instalación de MXNet.

Opción 1: SSH en su dispositivo y ejecutar scripts

1. SSH en su dispositivo.
2. Ejecute los siguientes scripts para comprobar que las dependencias están instaladas correctamente.

   ```
   sudo python3.7 -c "import mxnet; print(mxnet.__version__)"
   ```

   ```
   sudo python3.7 -c "import numpy; print(numpy.__version__)"
   ```

   El resultado imprime el número de versión y el script debe salir sin errores.

Opción 2: Ejecutar la prueba de dependencia IDT

1. Asegúrese de que `device.json` esté configurado para la cualificación de ML. Para obtener más información, consulte the section called “Configurar device.json para cualificación de ML”.
2. Ejecute la prueba de dependencias para el marco.

   ```
   devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id mxnet_dependency_check
   ```

   El resumen de la prueba muestra un resultado `PASSED` para `mldependencies`.

Instalar dependencias de TensorFlow

Las pruebas de cualificación IDT para este marco tienen las siguientes dependencias:

- Python 3.6 o Python 3.7.

> **Note**
>
> Si está utilizando Python 3.6, debe crear un enlace simbólico desde binarios de Python 3.7 a Python 3.6. Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass. Por ejemplo:
>
> ```
> sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
> ```

- TensorFlow 1.x.

Instalación de TensorFlow

Siga las instrucciones de la documentación de TensorFlow para instalar TensorFlow 1.x con pip o desde el origen.

> **Note**
>
> Si en el dispositivo están instalados Python 2.x y Python 3.x, use Python 3.x en los comandos que ejecute para instalar las dependencias.

Validación de la instalación de TensorFlow

Elija una de las siguientes opciones para validar la instalación de TensorFlow.

Opción 1: SSH en su dispositivo y ejecutar un script

1. SSH en su dispositivo.
2. Ejecute el siguiente script para comprobar que la dependencia está instalada correctamente.

   ```
   sudo python3.7 -c "import tensorflow; print(tensorflow.__version__)"
   ```

   El resultado imprime el número de versión y el script debe salir sin errores.

Opción 2: Ejecutar la prueba de dependencia IDT

1. Asegúrese de que `device.json` esté configurado para la cualificación de ML. Para obtener más información, consulte the section called “Configurar device.json para cualificación de ML”.
2. Ejecute la prueba de dependencias para el marco.

   ```
   devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id tensorflow_dependency_check
   ```

   El resumen de la prueba muestra un resultado `PASSED` para `mldependencies`.

Instalar las dependencias de Amazon SageMaker Neo Deep Learning Runtime (DLR)

Las pruebas de cualificación IDT para este marco tienen las siguientes dependencias:

- Python 3.6 o Python 3.7.

> **Note**
>
> Si está utilizando Python 3.6, debe crear un enlace simbólico desde binarios de Python 3.7 a Python 3.6. Esto configura su dispositivo para que cumpla con el requisito de Python para AWS IoT Greengrass. Por ejemplo:
>
> ```
> sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
> ```

- SageMaker Neo DLR.
- numpy.

Después de instalar las dependencias de prueba DLR, debe compilar el modelo.

Instalación de DLR

Siga las instrucciones de la documentación de DLR para instalar el DLR Neo.

> **Note**
>
> Si en el dispositivo están instalados Python 2.x y Python 3.x, use Python 3.x en los comandos que ejecute para instalar las dependencias.

Validación de la instalación de DLR

Elija una de las siguientes opciones para validar la instalación de DLR.

Opción 1: SSH en su dispositivo y ejecutar scripts

1. SSH en su dispositivo.
2. Ejecute los siguientes scripts para comprobar que las dependencias están instaladas correctamente.

   ```
   sudo python3.7 -c "import dlr; print(dlr.__version__)"
   ```

   ```
   sudo python3.7 -c "import numpy; print(numpy.__version__)"
   ```

   El resultado imprime el número de versión y el script debe salir sin errores.

Opción 2: Ejecutar la prueba de dependencia IDT

1. Asegúrese de que `device.json` esté configurado para la cualificación de ML. Para obtener más información, consulte the section called “Configurar device.json para cualificación de ML”.
2. Ejecute la prueba de dependencias para el marco.

   ```
   devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-
   id dlr_dependency_check
   ```

   El resumen de la prueba muestra un resultado `PASSED` para `mldependencies`.

Compilar el modelo DLR

Debe compilar el modelo DLR antes de poder usarlo para pruebas de cualificación de ML. Para los pasos, elija una de las siguientes opciones.

Opción 1: Utilizar Amazon SageMaker para compilar el modelo

Siga estos pasos para utilizar SageMaker para compilar el modelo de ML proporcionado por IDT. Este modelo está entrenado previamente con Apache MXNet.

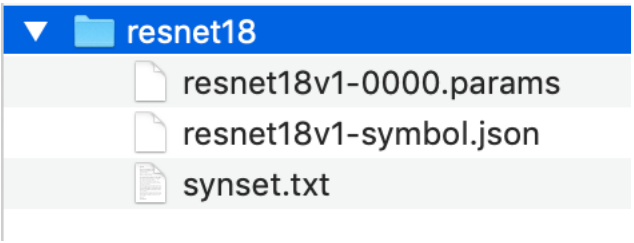
1. Compruebe que el tipo de dispositivo es compatible con SageMaker. Para obtener más información, consulte las opciones del dispositivo de destino en la Referencia de API de Amazon SageMaker. Si el tipo de dispositivo no es compatible actualmente con SageMaker, sigue los pasos descritos en the section called “Opción 2: Utilice TVM para compilar el modelo DLR”.

   > **Note**
   >
   > Ejecutar la prueba de DLR con un modelo compilado por SageMaker puede tardar 4 o 5 minutos. No detenga IDT durante este tiempo.

2. Descargue el archivo tarball que contiene el modelo MXNet no compilado y entrenado previamente para DLR:

   - dlr-noncompiled-model-1.0.tar.gz

3. Descomprima el tarball. Este comando genera la siguiente estructura de directorios.

4. Saque `synset.txt` del directorio `resnet18`. Anote la nueva ubicación. Copie este archivo en el directorio del modelo compilado más adelante.

5. Comprima el contenido del directorio `resnet18`.

   ```
   tar cvfz model.tar.gz resnet18v1-symbol.json resnet18v1-0000.params
   ```

6. Cargue el archivo comprimido en un bucket de Amazon S3 de su cuenta de Cuenta de AWS y, a continuación, siga los pasos que se indican en Compilar un modelo (consola) para crear un trabajo de compilación.

   a. En Configuración de entrada, utilice los siguientes valores:

      - En Configuración de entrada de datos, escriba `{"data": [1, 3, 224, 224]}`.

- En Marco de machine learning, elija `MXNet`.

b. En Configuración de salida, utilice los siguientes valores:

- En Ubicación de salida de S3, escriba la ruta de acceso al bucket de Amazon S3 o carpeta donde desea almacenar el modelo compilado.
- En Dispositivo de destino, elija el tipo de dispositivo.

7. Descargue el modelo compilado desde la ubicación de salida especificada y, a continuación, descomprima el archivo.
8. Copie `synset.txt` en el directorio del modelo compilado.
9. Cambie el nombre del directorio del modelo compilado a `resnet18`.

   El directorio del modelo compilado debe tener la siguiente estructura de directorios.

Opción 2: Utilice TVM para compilar el modelo DLR

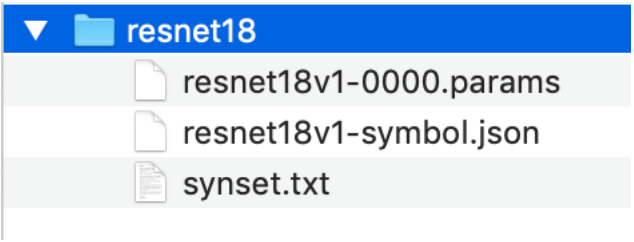
Siga estos pasos para utilizar TVM para compilar el modelo de ML proporcionado por IDT. Este modelo está entrenado previamente con Apache MXNet, por lo que debe instalar MXNet en el equipo o dispositivo donde se compila el modelo. Para instalar MXNet, siga las instrucciones de la documentación de MXNet.

> **Note**
>
> Le recomendamos que compile el modelo en el dispositivo de destino. Esta práctica es opcional, pero puede ayudar a garantizar la compatibilidad y mitigar posibles problemas.

1. Descargue el archivo tarball que contiene el modelo MXNet no compilado y entrenado previamente para DLR:

- dlr-noncompiled-model-1.0.tar.gz

2. Descomprima el tarball. Este comando genera la siguiente estructura de directorios.



3. Siga las instrucciones de la documentación de TVM para compilar e instalar TVM desde el origen para su plataforma.

4. Una vez compilado TVM, ejecute la compilación TVM para el modelo resnet18. Los siguientes pasos se basan en la explicación de inicio rápido para compilar modelos de aprendizaje profundo en la documentación de TVM.

   a. Abra el archivo `relay_quick_start.py` desde el repositorio de TVM clonado.

   b. Actualice el código que define una red neuronal en relé. Puede utilizar una de las siguientes opciones:

      - Opción 1: Utilice `mxnet.gluon.model_zoo.vision.get_model` para obtener el módulo de relé y los parámetros:

```
from mxnet.gluon.model_zoo.vision import get_model
block = get_model('resnet18_v1', pretrained=True)
mod, params = relay.frontend.from_mxnet(block, {"data": data_shape})
```

      - Opción 2: Desde el modelo no compilado que descargó en el paso 1, copie los siguientes archivos en el mismo directorio que el archivo `relay_quick_start.py`. Estos archivos contienen el módulo de relé y los parámetros.
        - `resnet18v1-symbol.json`
        - `resnet18v1-0000.params`

   c. Actualice el código que guarda y carga el módulo compilado para utilizar el siguiente código.

```
from tvm.contrib import util
path_lib = "deploy_lib.so"
#  Export the model library based on your device architecture
lib.export_library("deploy_lib.so", cc="aarch64-linux-gnu-g++")
with open("deploy_graph.json", "w") as fo:
```

```
    fo.write(graph)
with open("deploy_param.params", "wb") as fo:
    fo.write(relay.save_param_dict(params))
```

d. Compile el modelo:

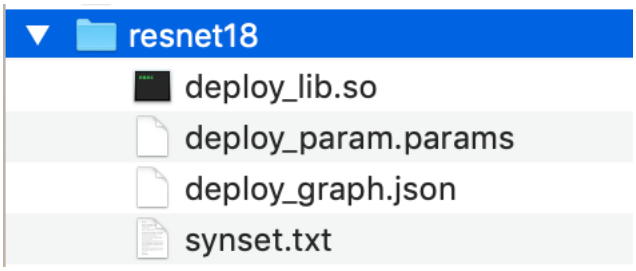
```
python3 tutorials/relay_quick_start.py --build-dir ./model
```

Este comando genera los archivos siguientes.

- `deploy_graph.json`
- `deploy_lib.so`
- `deploy_param.params`

5. Copie los archivos de modelo generados en un directorio denominado `resnet18`. Este es su directorio de modelo compilado.
6. Copie el directorio del modelo compilado en el equipo host. A continuación, copie `synset.txt` del modelo sin compilar que descargó en el paso 1 en el directorio del modelo compilado.

   El directorio del modelo compilado debe tener la siguiente estructura de directorios.



A continuación, configure las credenciales de AWS y el archivo `device.json`.

Configure los ajustes de IDT para ejecutar el conjunto de cualificación de AWS IoT Greengrass

Antes de ejecutar las pruebas, debe ajustar la configuración de las credenciales de AWS y los dispositivos en su equipo host.

Configuración de sus credenciales de AWS

Debe configurar sus credenciales de usuario de IAM en el archivo *<device-tester-extract-location>* `/configs/config.json`. Utilice las credenciales de IDT para el usuario de AWS IoT

Greengrass creado en the section called “Cree y configure un Cuenta de AWS”. Puede especificar sus credenciales de una de las dos formas siguientes:

- Archivo de credenciales
- Variables de entorno

Configuración de las credenciales de AWS con un archivo de credenciales

IDT utiliza el mismo archivo de credenciales que la AWS CLI. Para obtener más información, consulte Archivos de configuración y credenciales.

La ubicación del archivo de credenciales varía en función del sistema operativo que utilice:

- macOS, Linux: `~/.aws/credentials`
- Windows: `C:\Users\`*`UserName`*`\.aws\credentials`

Añada sus credenciales de AWS al archivo de `credentials` con el siguiente formato:

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

Para configurar IDT para AWS IoT Greengrass con el fin de usar las credenciales de AWS desde su archivo de `credentials`, edite su archivo `config.json` tal y como se indica a continuación:

```
{
 "awsRegion": "us-west-2",
 "auth": {
  "method": "file",
  "credentials": {
   "profile": "default"
  }
 }
}
```

> **Note**
>
> Si no utiliza el perfil `default` de AWS asegúrese de cambiar el nombre de perfil en su archivo `config.json`. Para obtener más información, consulte Perfiles con nombre.

Configuración de las credenciales de AWS con variables de entorno

Las variables de entorno son las variables que mantiene el sistema operativo y utilizan los comandos del sistema. No se guardan si cierra la sesión de SSH. IDT para AWS IoT Greengrass puede utilizar las variables de entorno `AWS_ACCESS_KEY_ID` y `AWS_SECRET_ACCESS_KEY` para almacenar sus credenciales de AWS.

Para establecer estas variables en Linux, MacOS, o Unix, utilice export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Para establecer estas variables en Windows, utilice set:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Para configurar IDT para utilizar las variables de entorno, edite la sección `auth` de su archivo `config.json`. A continuación se muestra un ejemplo:

```
{
 "awsRegion": "us-west-2",
 "auth": {
  "method": "environment"
 }
}
```

Configurar device.json

Además de las credenciales de AWS, IDT para AWS IoT Greengrass requiere información acerca de los dispositivos en los que se ejecutan pruebas (por ejemplo, dirección IP, información de inicio de sesión, sistema operativo y arquitectura de CPU).

Debe proporcionar esta información utilizando la plantilla `device.json` ubicada en *<device_tester_extract_location>*`/configs/device.json`:

Physical device

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "container",
        "value": "yes | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "yes | no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ********** Remove the section below if the device is not qualifying for ML
 **************,
      {
        "name": "mlLambdaContainerizationMode",
        "value": "container | process | both"
```

```
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      },

*******************************************************************************************
   ],
   ********** Remove the section below if the device is not qualifying for HSI
***************
   "hsm": {
     "p11Provider": "/path/to/pkcs11ProviderLibrary",
     "slotLabel": "<slot_label>",
     "slotUserPin": "<slot_pin>",
     "privateKeyLabel": "<key_label>",
     "openSSLEngine": "/path/to/openssl/engine"
   },

*******************************************************************************************
   ********** Remove the section below if the device is not qualifying for ML
***************
   "machineLearning": {
     "dlrModelPath": "/path/to/compiled/dlr/model",
     "environmentVariables": [
       {
         "key": "<environment-variable-name>",
         "value": "<Path:$PATH>"
       }
     ],
     "deviceResources": [
       {
         "name": "<resource-name>",
         "path": "<resource-path>",
         "type": "device | volume"
       }
     ]
   },

*******************************************************************************************
   "kernelConfigLocation": "",
   "greengrassLocation": "",
   "devices": [
     {
       "id": "<device-id>",
```

```
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

> **Note**
>
> Especifique `privKeyPath` solo si `method` está establecido en `pki`
> Especifique `password` solo si `method` está establecido en `password`

Docker container

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64"
      },
      {
        "name": "container",
        "value": "no"
```

```
      },
      {
        "name": "docker",
        "value": "no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ********** Remove the section below if the device is not qualifying for ML
**************,
      {
        "name": "mlLambdaContainerizationMode",
        "value": "process"
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      },

*********************************************************************************************
    ],
    ********** Remove the section below if the device is not qualifying for ML
***************
    "machineLearning": {
      "dlrModelPath": "/path/to/compiled/dlr/model",
      "environmentVariables": [
        {
          "key": "<environment-variable-name>",
          "value": "<Path:$PATH>"
        }
      ],
      "deviceResources": [
        {
          "name": "<resource-name>",
          "path": "<resource-path>",
```

```
          "type": "device | volume"
        }
      ]
    },

 ********************************************************************************************
    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "docker",
          "containerId": "<container-name | container-id>",
          "containerUser": "<user>"
        }
      }
    ]
  }
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID alfanumérico definido por el usuario que identifica de forma única una colección de dispositivos que se conoce como grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben tener idéntico hardware. Al ejecutar un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo. Se utilizan varios dispositivos para ejecutar diferentes pruebas.

`sku`

Un valor alfanumérico que identifica de forma única el dispositivo a prueba. El SKU se utiliza para realizar un seguimiento de placas cualificadas.

> **Note**
>
> Si desea enumerar la placa en el catálogo de dispositivos de AWS Partner, el SKU que especifique aquí debe coincidir con el SKU que utilice en el proceso de publicación.

`features`

Una matriz que contenga las características compatibles del dispositivo. Todas las características son obligatorias.

`os` y `arch`

Combinaciones de sistemas operativos (SO) compatibles:

- `linux`, `x86_64`
- `linux`, `armv6l`
- `linux`, `armv7l`
- `linux`, `aarch64`
- `ubuntu`, `x86_64`
- `openwrt`, `armv7l`
- `openwrt`, `aarch64`

> **Note**
>
> Si usa IDT para probar la ejecución de AWS IoT Greengrass en un contenedor de Docker, solo se admite la arquitectura Docker x86_64.

`container`

Valida que el dispositivo cumple todos los requisitos de software y hardware para ejecutar funciones de Lambda en modo contenedor en un núcleo de Greengrass.

El valor válido es `yes` o `no`.

`docker`

Valida que el dispositivo cumple todas las dependencias técnicas necesarias para utilizar el conector de implementación de aplicaciones de Greengrass Docker para ejecutar contenedores.

El valor válido es `yes` o `no`.

`streamManagement`

Valida que el dispositivo cumple todas las dependencias técnicas necesarias para ejecutar el administrador de transmisiones de AWS IoT Greengrass.

El valor válido es `yes` o `no`.

`hsi`

Verifica que la biblioteca compartida de HSI proporcionada pueda interactuar con el módulo de seguridad de hardware (HSM) y se implementan correctamente las API de PKCS#11 necesarias. La biblioteca HSM y compartida debe poder firmar una CSR, realizar las operaciones de TLS y proporcionar las longitudes de clave y el algoritmo de clave pública correctos.

El valor válido es `yes` o `no`.

`ml`

Valida que el dispositivo cumpla todas las dependencias técnicas necesarias para realizar la inferencia de ML localmente.

El valor válido puede ser cualquier combinación de `mxnet`, `tensorflow`, `dlr` y `no` (por ejemplo, `mxnet`, `mxnet,tensorflow`, `mxnet,tensorflow,dlr` o `no`).

`mlLambdaContainerizationMode`

Valida que el dispositivo cumple todas las dependencias técnicas necesarias para realizar la inferencia ML en modo contenedor en un dispositivo Greengrass.

El valor válido es `container`, `process` o `both`.

`processor`

Valida que el dispositivo cumpla con todos los requisitos de hardware del tipo de procesador especificado.

El valor válido es `cpu` o `gpu`.

> **Note**
>
> Si no quiere utilizar la característica `container`, `docker`, `streamManager`, `hsi` o `ml`, puede establecer el valor correspondiente `value` a `no`.
> Docker solo admite la calificación de características para `streamManagement` y `ml`.

`machineLearning`

Opcional. Información de configuración para pruebas de cualificación ML. Para obtener más información, consulte the section called “Configurar device.json para cualificación de ML”.

`hsm`

Opcional. Contiene información de configuración para realizar pruebas con un módulo de seguridad de hardware (HSM) de AWS IoT Greengrass. De lo contrario, la propiedad `hsm` debe omitirse. Para obtener más información, consulte Integración de la seguridad de hardware.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`hsm.p11Provider`

La ruta absoluta a la biblioteca que puede cargar libdl de la implementación de PKCS#11.

`hsm.slotLabel`

La etiqueta de ranura que se utiliza para identificar el módulo de hardware.

`hsm.slotUserPin`

El PIN de usuario que se utiliza para autenticar el núcleo AWS IoT Greengrass en el módulo.

`hsm.privateKeyLabel`

Es la etiqueta que se utiliza para identificar la clave en el módulo de hardware.

`hsm.openSSLEngine`

La ruta absoluta al archivo `.so` del motor de OpenSSL que habilita la compatibilidad con PKCS#11 en OpenSSL. La utiliza el agente de actualización OTA de AWS IoT Greengrass.

`devices.id`

Un identificador único y definido por el usuario para el dispositivo que se está probando.

`connectivity.protocol`

El protocolo de comunicación que se usará para la comunicación con este dispositivo. Actualmente, los únicos valores que se admiten son `ssh` para dispositivos físicos y `docker` para contenedores de Docker.

`connectivity.ip`

La dirección IP del dispositivo que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.containerId`

El ID de contenedor o el nombre del contenedor de Docker que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `docker`.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.password`

La contraseña se utiliza para iniciar sesión en el dispositivo que se va a probar.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo que se está probando.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`connectivity.auth.credentials.user`

El nombre de usuario para iniciar sesión en el dispositivo que se está probando.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo que se está probando.

`connectivity.port`

Opcional. El número de puerto a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`greengrassLocation`

La ubicación del software AWS IoT Greengrass Core en sus dispositivos.

En los dispositivos físicos, este valor solo se utiliza cuando se utiliza una instalación existente de AWS IoT Greengrass. Utilice este atributo para indicar a IDT que utilice la versión del software AWS IoT Greengrass Core instalado en sus dispositivos.

Al ejecutar pruebas en un contenedor de Docker a partir de una imagen de Docker o un Dockerfile proporcionado por AWS IoT Greengrass, establezca este valor en `/greengrass`.

`kernelConfigLocation`

Opcional. La ruta al archivo de configuración del kernel. AWS IoT Device Tester utiliza este archivo para comprobar si los dispositivos tienen habilitadas las características de kernel requeridas. Si no se especifica, IDT utiliza las siguientes rutas para buscar el archivo de configuración de kernel: `/proc/config.gz` y `/boot/config-`*`<kernel-version>`*. AWS IoT Device Tester utiliza la primera ruta que encuentra.

Configurar device.json para cualificación de ML

En esta sección se describen las propiedades opcionales del archivo de configuración del dispositivo que se aplican a la cualificación de ML. Si tiene previsto ejecutar pruebas para la cualificación de ML, debe definir las propiedades que se aplican a su caso de uso.

Puede utilizar la plantilla `device-ml.json` para definir los ajustes de configuración del dispositivo. Esta plantilla contiene las propiedades de ML opcionales. También puede usar `device.json` y agregar las propiedades de cualificación de ML. Estos archivos se encuentran en *`<device-tester-extract-location>`*`/configs` e incluyen propiedades de cualificación de ML. Si utiliza `device-ml.json`, debe cambiar el nombre del archivo a `device.json` antes de ejecutar pruebas de IDT.

Para obtener información acerca de las propiedades de configuración de dispositivos que no se aplican a la cualificación de ML, consulte the section called “Configurar device.json”.

`ml` en la matriz `features`

Los marcos de ML que admite la placa. Esta propiedad requiere IDT v3.1.0 o una versión posterior.

- Si la placa solo admite un marco, especifíquelo. Por ejemplo:

```
{
    "name": "ml",
    "value": "mxnet"
}
```

- Si la placa admite varios marcos, especifique los marcos como lista separada por comas. Por ejemplo:

```
{
    "name": "ml",
    "value": "mxnet,tensorflow"
}
```

`mlLambdaContainerizationMode` en la matriz `features`

El modo de creación de contenedores con el que desea probar. Esta propiedad requiere IDT v3.1.0 o una versión posterior.

- Elija `process` para ejecutar el código de inferencia de ML con una función de Lambda no en contenedor. Esta opción requiere AWS IoT Greengrass v1.10.x o posterior.
- Elija `container` para ejecutar el código de inferencia ML con una función de Lambda en contenedor.
- Elija `both` para ejecutar el código de inferencia ML con ambos modos. Esta opción requiere AWS IoT Greengrass v1.10.x o posterior.

`processor` en la matriz `features`

Indica el acelerador de hardware compatible con la placa. Esta propiedad requiere IDT v3.1.0 o una versión posterior.

- Elija `cpu` si la placa utiliza una CPU como procesador.
- Elija `gpu` si la placa utiliza una GPU como procesador.

`machineLearning`

Opcional. Información de configuración para pruebas de cualificación ML. Esta propiedad requiere IDT v3.1.0 o una versión posterior.

`dlrModelPath`

Necesario para usar el marco `dlr`. La ruta absoluta al directorio de modelo compilado DLR, que debe haberse denominado `resnet18`. Para obtener más información, consulte the section called “Compilar el modelo DLR”.

> **Note**
>
> A continuación, se muestra una ruta de ejemplo en macOS: `/Users/<user>/Downloads/resnet18`.

`environmentVariables`

Una matriz de pares clave-valor que puede pasar dinámicamente la configuración a las pruebas de inferencia de ML. Opcional para dispositivos de CPU. Puede usar esta sección para agregar variables de entorno específicas del marco requeridas por el tipo de dispositivo. Para obtener información sobre estos requisitos, consulte el sitio web oficial del marco o el dispositivo. Por ejemplo, para ejecutar pruebas de inferencia de MXNet en algunos dispositivos, pueden ser necesarias las siguientes variables de entorno.

```
"environmentVariables": [
    ...
    {
        "key": "PYTHONPATH",
        "value": "$MXNET_HOME/python:$PYTHONPATH"
    },
    {
        "key": "MXNET_HOME",
        "value": "$HOME/mxnet/"
    },
    ...
]
```

> **Note**
>
> El campo `value` puede variar en función de la instalación de MXNet.

Si está probando funciones de Lambda que se ejecutan con creación de contenedores en dispositivos GPU, agregue variables de entorno para la biblioteca de GPU. Esto permite que la GPU realice cálculos. Para utilizar diferentes bibliotecas de GPU, consulte la documentación oficial de la biblioteca o dispositivo.

> **Note**
>
> Configure las siguientes claves si la característica `mlLambdaContainerizationMode` está establecida en `container` o `both`.

```
"environmentVariables": [
    {
        "key": "PATH",
        "value": "<path/to/software/bin>:$PATH"
    },
    {
        "key": "LD_LIBRARY_PATH",
        "value": "<path/to/ld/lib>"
    },
    ...
]
```

`deviceResources`

Requerido por los dispositivos de GPU. Contiene recursos locales a los que se puede acceder mediante funciones de Lambda. Utilice esta sección para agregar recursos de volumen y dispositivos locales.

- Para recursos de dispositivo, especifique `"type": "device"`. Para los dispositivos GPU, los recursos del dispositivo deben ser archivos de dispositivo relacionados con la GPU bajo `/dev`.

> **Note**
>
> El directorio `/dev/shm` es una excepción. Se puede configurar solo como un recurso de volumen.

- Para recursos de volumen, especifique `"type": "volume"`.

Ejecute el conjunto de cualificación de AWS IoT Greengrass

Después de configurar los ajustes necesarios, puede iniciar las pruebas. El tiempo de ejecución del conjunto completo de pruebas depende de su hardware. Como referencia, se tarda aproximadamente 30 minutos en completar el conjunto de pruebas completo en una unidad Raspberry Pi 3B.

Los comandos `run-suite` de ejemplo siguientes muestran cómo ejecutar las pruebas de cualificación para un grupo de dispositivos. Un grupo de dispositivos es un conjunto de dispositivos idénticos.

IDT v3.0.0 and later

Ejecute todos los grupos de prueba en un conjunto de pruebas especificado.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --pool-
id <pool-id>
```

Utilice el `list-suites` comando para enumerar los conjuntos de pruebas que se encuentran en la carpeta `tests`.

Ejecute un grupo de pruebas específico en un conjunto de pruebas.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --group-
id <group-id> --pool-id <pool-id>
```

Utilice el comando `list-groups` para enumerar los grupos de prueba en un conjunto de pruebas.

Ejecute un caso de prueba específico en un grupo de prueba.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-
id <test-id>
```

Ejecute varios casos de prueba en un grupo de prueba.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-
id <test-id1>,<test-id2>
```

Enumere los casos de prueba en un grupo de prueba.

```
devicetester_[linux | mac | win_x86-64] list-test-cases --group-id <group-id>
```

Las opciones del comando `run-suite` son opcionales. Por ejemplo, puede omitir `pool-id` solo si tiene un grupo de dispositivos definido en el archivo `device.json`. O bien, puede omitir `suite-id` si desea ejecutar la última versión del conjunto de pruebas en la carpeta `tests`.

> **Note**
>
> IDT le pregunta si está disponible en línea una versión más reciente del conjunto de pruebas. Para obtener más información, consulte the section called “Establezca el comportamiento de actualización predeterminado”.

Para obtener más información acerca de `run-suite` y otros comandos IDT, consulte the section called “Comandos de IDT”.

IDT v2.3.0 and earlier

Ejecute todos los grupos de prueba en un conjunto especificado.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --pool-
id <pool-id>
```

Ejecute un grupo de prueba específico.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --group-
id <group-id> --pool-id <pool-id>
```

`suite-id` y `pool-id` son opcionales si ejecuta un único conjunto de pruebas en un único grupo de dispositivos. Esto significa que solo tiene un grupo de dispositivos definido en su archivo `device.json`.

Compruebe las dependencias de Greengrass

Le recomendamos que ejecute el grupo de pruebas del comprobador de dependencias para asegurarse de que todas las dependencias de Greengrass están instaladas antes de ejecutar grupos de pruebas relacionados. Por ejemplo:

- Ejecute `ggcdependencies` antes de ejecutar los grupos de prueba de cualificación del núcleo.
- Ejecute `containerdependencies` antes de ejecutar grupos de prueba específicos de contenedor.
- Ejecute `dockerdependencies` antes de ejecutar grupos de prueba específicos de Docker.
- Ejecute `ggcstreammanagementdependencies` antes de ejecutar los grupos de prueba específicos del administrador de secuencias.

Establezca el comportamiento de actualización predeterminado

Al iniciar una ejecución de prueba, IDT comprueba en línea una versión más reciente del conjunto de pruebas. Si hay alguna disponible, IDT le pedirá que actualice a la última versión disponible. Puede establecer la marca `upgrade-test-suite` (o u) para controlar el comportamiento de actualización predeterminado. Los valores válidos son:

- y. IDT descarga y utiliza la última versión disponible.
- n (predeterminada). IDT utiliza la versión especificada en la opción `suite-id`. Si no se especifica `suite-id`, IDT utiliza la versión más reciente en la carpeta `tests`.

Si no incluye la marca `upgrade-test-suite`, IDT le preguntará cuándo hay una actualización disponible y esperará 30 segundos a la entrada (y o n). Si no se introduce ninguna entrada, el valor predeterminado es n y continúa ejecutando las pruebas.

Los siguientes ejemplos muestran casos de uso comunes para esta característica:

Utilizar automáticamente las últimas pruebas disponibles para un grupo de pruebas.

```
devicetester_linux run-suite -u y --group-id mqtt --pool-id DevicePool1
```

Ejecutar pruebas en una versión específica del conjunto de pruebas.

```
devicetester_linux run-suite -u n --suite-id GGQ_1.0.0 --group-id mqtt --pool-id
 DevicePool1
```

Solicitar actualizaciones en tiempo de ejecución.

```
devicetester_linux run-suite --pool-id DevicePool1
```

Comandos de IDT para AWS IoT Greengrass

Los comandos IDT se encuentran en el directorio *<device-tester-extract-location>*/bin. Úselos para las siguientes operaciones:

IDT v3.0.0 and later

`help`

Enumera información acerca del comando especificado.

`list-groups`

Muestra los grupos de un conjunto de prueba determinado.

`list-suites`

Muestra los conjuntos de prueba disponibles.

`list-supported-products`

Enumera los productos compatibles, en este caso las versiones de AWS IoT Greengrass y las versiones del conjunto de pruebas para la versión actual de IDT.

`list-test-cases`

Enumera los casos de prueba en un grupo de prueba determinado. Se admite la siguiente opción:

- `group-id`. El grupo de prueba que se va a buscar. Esta opción es necesaria y debe especificar un solo grupo.

`run-suite`

Ejecuta un conjunto de pruebas en un grupo de dispositivos. Las siguientes son algunas de las opciones admitidas:

- `suite-id`. La versión del conjunto de pruebas que se va a ejecutar. Si no se especifica, IDT utiliza la versión más reciente de la carpeta `tests`.
- `group-id`. Los grupos de prueba que se van a ejecutar, como una lista separada por comas. Si no se especifica, IDT ejecuta todos los grupos de prueba del conjunto de pruebas.
- `test-id`. Los casos de prueba a ejecutar, como lista separada por comas. Cuando se especifique, `group-id` debe especificar un solo grupo.
- `pool-id`. El grupo de dispositivos que se va a probar. Debe especificar un grupo si tiene varios grupos de dispositivos definidos en el archivo `device.json`.
- `upgrade-test-suite`. Controla cómo se manejan las actualizaciones de la versión del conjunto de pruebas. A partir de IDT v3.0.0, IDT comprueba en línea las versiones actualizadas del conjunto de pruebas. Para obtener más información, consulte the section called “Versiones del conjunto de pruebas”.
- `stop-on-first-failure`. Configura IDT para detener la ejecución en el primer error. Esta opción debe utilizarse con `group-id` para depurar los grupos de prueba especificados. No utilice esta opción cuando ejecute un conjunto de pruebas completo para generar un informe de cualificación.
- `update-idt`. Establece la respuesta del mensaje para actualizar IDT. `Y` si la entrada detiene la ejecución de la prueba si IDT detecta que hay una versión más reciente. `N` si la entrada continúa la ejecución de la prueba.
- `update-managed-policy`. `Y` si que la entrada detiene la ejecución de la prueba si IDT detecta que la política administrada del usuario no está actualizada. `N` si la entrada continúa la ejecución de la prueba.

Para obtener más información acerca de `run-suite` las opciones, utilice la opción `help`:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v2.3.0 and earlier

`help`

Enumera información acerca del comando especificado.

`list-groups`

Muestra los grupos de un conjunto de prueba determinado.

`list-suites`

Muestra los conjuntos de prueba disponibles.

`run-suite`

Ejecuta un conjunto de pruebas en un grupo de dispositivos.

Para obtener más información acerca de `run-suite` las opciones, utilice la opción `help`:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

Descripción de los resultados y de los registros

En esta sección se describe cómo ver e interpretar registros e informes de resultados de IDT.

Ver los resultados

Mientras ejecuta, IDT escribe errores en la consola, en archivos de registro y en informes de prueba. Una vez que IDT completa el conjunto de pruebas de cualificación, genera dos informes de prueba. Estos informes se pueden encontrar en *<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Ambos informes capturan los resultados de la ejecución del conjunto de pruebas de cualificación.

El `awsiotdevicetester_report.xml` es el informe de prueba de cualificación que envía a AWS para mostrar su dispositivo en el AWS Partner Partner Device Catalog. El informe contiene los componentes siguientes:

- La versión de IDT.
- La versión AWS IoT Greengrass que se ha probado.
- El SKU y el grupo de dispositivos especificado en el archivo `device.json`.

- Las características del grupo de dispositivos especificado en el archivo `device.json`.
- El resumen de agregación de los resultados de las pruebas.
- Un desglose de los resultados de las pruebas por bibliotecas que se probaron en función de las características de los dispositivos (por ejemplo, acceso a recursos locales, shadow, MQTT, etc.).

El informe `GGQ_Result.xml` está en formato XML JUnit. Puede integrarlo en plataformas de integración/implementación continua como Jenkins, Bamboo, etc. El informe contiene los componentes siguientes:

- Resumen de agregación de los resultados de pruebas.
- Desglose de resultados de pruebas por funcionalidad de AWS IoT Greengrass probada.

Interpretación de los informes de IDT

La sección de informe en `awsiotdevicetester_report.xml` o `awsiotdevicetester_report.xml` enumera las pruebas que se ejecutaron y los resultados.

La primera etiqueta XML `<testsuites>` contiene el resumen de la ejecución de las pruebas. Por ejemplo:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Atributos que se utilizan en la etiqueta **`<testsuites>`**

`name`

El nombre del grupo de prueba.

`time`

El tiempo, en segundos, que se ha tardado en ejecutar el conjunto de cualificación.

`tests`

El número de pruebas ejecutadas.

`failures`

El número de pruebas que se ejecutaron, pero que no se superaron.

`errors`

El número de pruebas que IDT no ha podido ejecutar.

`disabled`

Este atributo no se utiliza y se puede omitir.

El archivo `awsiotdevicetester_report.xml` contiene una etiqueta `<awsproduct>` que tiene información sobre el producto que se está probando y las características del producto que se han validado después de ejecutar un conjunto de pruebas.

Atributos que se utilizan en la etiqueta **`<awsproduct>`**

`name`

El nombre del producto que se está probando.

`version`

La versión del producto que se está probando.

`features`

Las características validadas. Las características marcadas como `required` son necesarias para solicitar la cualificación de la placa. En el siguiente fragmento se muestra cómo aparece esta información en el archivo `awsiotdevicetester_report.xml`:

```
<feature name="aws-iot-greengrass-no-container" value="supported" type="required"></
feature>
```

Las características marcadas como `optional` no son necesarias para la cualificación. Los siguientes fragmentos muestran características opcionales:

```
<feature name="aws-iot-greengrass-container" value="supported" type="optional"></
feature>

<feature name="aws-iot-greengrass-hsi" value="not-supported" type="optional"></
feature>
```

Si no hay errores de pruebas para las características requeridas, el dispositivo cumple los requisitos técnicos para ejecutar AWS IoT Greengrass y puede interoperar con servicios de AWS IoT. Si quiere

mostrar su dispositivo en el AWS Partner Partner Device Catalog, puede utilizar este informe como prueba de cualificación.

Si se producen errores en pruebas, puede identificar la prueba fallido revisando las etiquetas XML `<testsuites>`. Las etiquetas XML `<testsuite>` dentro de la etiqueta `<testsuites>` muestran el resumen del resultado de la prueba de un grupo de prueba. Por ejemplo:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

El formato es similar a la etiqueta `<testsuites>`, pero con un atributo `skipped` que no se utiliza y que se puede pasar por alto. Dentro de cada etiqueta XML `<testsuite>`, hay etiquetas `<testcase>` para cada prueba ejecutada para un grupo de prueba. Por ejemplo:

```
<testcase classname="Security Combination (IPD + DCM) Test Context" name="Security
 Combination IP Change Tests sec4_test_1: Should rotate server cert when IPD disabled
 and following changes are made:Add CIS conn info and Add another CIS conn info"
 attempts="1"></testcase>>
```

Atributos que se utilizan en la etiqueta **`<testcase>`**

`name`

El nombre de la prueba.

`attempts`

El número de veces que IDT ha ejecutado el caso de prueba.

Cuando una prueba genera un error o si se produce un error, las etiquetas `<failure>` o `<error>` se añaden a la etiqueta `<testcase>` con información para la resolución de problemas. Por ejemplo:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

Visualización de registros de

IDT genera registros a partir de la ejecución de pruebas en *`<devicetester-extract-location>`*`/results/`*`<execution-id>`*`/logs`. Se generan dos conjuntos de registros:

`test_manager.log`

Registros generados a partir del componente Test Manager de AWS IoT Device Tester (por ejemplo, registros relacionados con la configuración, secuenciación de pruebas y generación de informes).

`<test_case_id>``.log (for example, ota.log)`

Son los registros del grupo de pruebas, incluidos los registros del dispositivo bajo prueba. Cuando una prueba devuelve un error, se crea un archivo tar.gz que contiene los registros del dispositivo sometido a prueba para la prueba (por ejemplo, `ota_prod_test_1_ggc_logs.tar.gz`).

Para obtener más información, consulte Solución de problemas de IDT para AWS IoT Greengrass.

Utilice IDT para desarrollar y ejecutar sus propios conjuntos de pruebas

A partir de la versión 4.0.0 de IDT, IDT para AWS IoT Greengrass combina una configuración y un formato de resultados estandarizados con un entorno de conjuntos de pruebas que le permite desarrollar conjuntos de pruebas personalizados para sus dispositivos y su software. Puede añadir pruebas personalizadas para su propia validación interna o proporcionárselas a sus clientes para la verificación de los dispositivos.

Utilice IDT para desarrollar y ejecutar conjuntos de pruebas personalizados, de la siguiente manera:

Para desarrollar conjuntos de pruebas personalizados

- Cree conjuntos de pruebas con lógica de prueba personalizada para el dispositivo Greengrass que desee probar.
- Proporcione a IDT sus conjuntos de pruebas personalizados para los corredores de pruebas. Incluya información sobre las configuraciones de configuración específicas de sus conjuntos de pruebas.

Para ejecutar conjuntos de pruebas personalizados

- Configure el dispositivo que desea probar.
- Implemente las configuraciones requeridas por los conjuntos de pruebas que desee utilizar.
- Utilice IDT para ejecutar sus conjuntos de pruebas personalizados.
- Vea los resultados de las pruebas y los registros de ejecución de las pruebas realizadas por IDT.

Descargar la última versión de AWS IoT Device Tester para AWS IoT Greengrass

Descargue la última versión de IDT y extraiga el software en una ubicación de su sistema de archivos en la que tenga permisos de lectura y escritura.

> **Note**
>
> IDT no admite la ejecución por parte de varios usuarios desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Le recomendamos que extraiga el paquete IDT en una unidad local y ejecute el binario IDT en su estación de trabajo local.
> Windows tiene una limitación de longitud de ruta de 260 caracteres. Si utiliza Windows, extraiga IDT en un directorio raíz como `C:\` o `D:\` para mantener las rutas por debajo del límite de 260 caracteres.

Flujo de trabajo de creación de conjuntos de prueba

Los conjuntos de pruebas se componen de tres tipos de archivos:

- Archivos de configuración JSON que proporcionan a IDT información sobre cómo ejecutar el conjunto de pruebas.
- Archivos ejecutables de prueba que IDT utiliza para ejecutar los casos de prueba.
- Se requieren archivos adicionales para ejecutar las pruebas.

Complete los siguientes pasos básicos para crear pruebas de IDT personalizadas:

1. Cree archivos de configuración JSON para su conjunto de pruebas.
2. Cree ejecutables de casos de prueba que contengan la lógica de prueba de su conjunto de pruebas.
3. Verifique y documente la información de configuración necesaria para que los ejecutores de pruebas ejecuten el conjunto de pruebas.
4. Compruebe que IDT pueda ejecutar su conjunto de pruebas y producir los resultados de las pruebas según lo esperado.

Para crear rápidamente un conjunto personalizado de muestra y ejecutarlo, siga las instrucciones que se indican en Tutorial: crear y ejecutar el ejemplo del conjunto de pruebas de IDT.

Para empezar a crear un conjunto de pruebas personalizado en Python, consulte Tutorial: Desarrollar un conjunto de pruebas de IDT sencillo.

Tutorial: crear y ejecutar el ejemplo del conjunto de pruebas de IDT

La descarga de AWS IoT Device Tester incluye el código fuente de un conjunto de pruebas de muestra. Puede completar este tutorial para crear y ejecutar el conjunto de pruebas de ejemplo y comprender cómo puede utilizar AWS IoT Device Tester para AWS IoT Greengrass y ejecutar conjuntos de pruebas personalizados.

En este tutorial, completará los siguientes pasos:

1. Cree el conjunto de pruebas de muestra
2. Use IDT para ejecutar el conjunto de pruebas de muestra

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Requisitos del equipo host
  - Última versión de Device Tester AWS IoT
  - Python 3.7 o posterior

    Para comprobar la versión de Python instalada en su ordenador, ejecute el siguiente comando:

    ```
    python3 --version
    ```

    En Windows, si el uso de este comando devuelve un error, utilice `python --version` en su lugar. Si el número de versión devuelto es 3.7 o superior, ejecute el siguiente comando en una terminal de Powershell para configurar `python3` como alias para el comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

Si no devuelve información sobre la versión o si la versión es inferior a 3.7, siga las instrucciones en Descargar Python para instalar Python 3.7+. Para obtener más información, consulte la documentación de Python.

- urllib3

  Para comprobar que `urllib3` se ha instalado correctamente, ejecute el siguiente comando:

  ```
  python3 -c 'import urllib3'
  ```

  Si `urllib3` no está instalado, ejecute el siguiente comando para instalarlo:

  ```
  python3 -m pip install urllib3
  ```

- Requisitos de los dispositivos
  - Un dispositivo con un sistema operativo Linux y una conexión de red a la misma red que el equipo host.

    Le recomendamos que utilice un Raspberry Pi con el sistema operativo Raspberry Pi. Debe configurar SSH en su Raspberry Pi para conectarse de forma remota.

Configure la información del dispositivo para IDT

Configure la información del dispositivo para que IDT ejecute la prueba. Debe actualizar la plantilla `device.json` ubicada en la carpeta *<device-tester-extract-location>*`/configs` con la siguiente información.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
```

```
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

En el objeto `devices`, proporcione la siguiente información:

`id`

Un identificador único definido por el usuario para su dispositivo.

`connectivity.ip`

La dirección IP de su dispositivo.

`connectivity.port`

Opcional. El número de puerto que se va a utilizar para las conexiones SSH al dispositivo.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.user`

El nombre de usuario utilizado para iniciar sesión en el dispositivo.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada utilizada para iniciar sesión en su dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`devices.connectivity.auth.credentials.password`

La contraseña utilizada para iniciar sesión en su dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

> **Note**
>
> Especifique `privKeyPath` solo si `method` está establecido en `pki`
> Especifique `password` solo si `method` está establecido en `password`

Cree el conjunto de pruebas de muestra

La carpeta *<device-tester-extract-location>*`/samples/python` contiene ejemplos de archivos de configuración, código fuente y el SDK de cliente de IDT que puede combinar en un conjunto de pruebas mediante los scripts de compilación proporcionados. El siguiente árbol de directorios muestra la ubicación de estos archivos de ejemplo:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
```

```
    ### ...
    ### python
        ### idt_client
```

Para crear el conjunto de pruebas, ejecute los siguientes comandos en el equipo host:

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```

Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

Esto crea el conjunto de pruebas de muestra en la carpeta `IDTSampleSuitePython_1.0.0` dentro de la carpeta *<device-tester-extract-location>*`/tests`. Revise los archivos de la carpeta `IDTSampleSuitePython_1.0.0` para comprender cómo está estructurado el conjunto de pruebas de muestra y consulte varios ejemplos de ejecutables de casos de prueba y archivos JSON de configuración de pruebas.

Siguiente paso: use IDT para ejecutar el conjunto de pruebas de muestra que creó.

Use IDT para ejecutar el conjunto de pruebas de muestra

Para ejecutar el conjunto de pruebas, ejecute los siguientes comandos en el equipo host:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT ejecuta el conjunto de pruebas de muestra y transmite los resultados a la consola. Cuando la prueba termine de ejecutarse, verá la siguiente información:

```
========== Test Summary ==========
Execution Time:         5s
Tests Completed:        4
Tests Passed:           4
Tests Failed:           0
Tests Skipped:          0
```

```
---------------------------------
Test Groups:
    sample_group:       PASSED
---------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

Solución de problemas

Utilice la siguiente información para resolver cualquier problema que pueda surgir al completar el tutorial.

El caso de prueba no se ejecuta correctamente

Si la prueba no se ejecuta correctamente, IDT transmite los registros de errores a la consola para ayudarle a solucionar los problemas de la prueba. Asegúrese de que cumple con todos los requisitos previos de este tutorial.

No se puede conectar al dispositivo que se está probando

Compruebe lo siguiente:

- El archivo `device.json` contiene la dirección IP, el puerto y la información de autenticación correctos.
- Puede conectarse a su dispositivo a través de SSH desde su equipo host.

Tutorial: Desarrollar un conjunto de pruebas de IDT sencillo

Un conjunto de pruebas combina lo siguiente:

- Ejecutables de prueba que contienen la lógica de prueba
- Archivos de configuración JSON que describen el conjunto de pruebas

Este tutorial le muestra cómo usar IDT para AWS IoT Greengrass para desarrollar un conjunto de pruebas de Python que contenga un único caso de prueba. En este tutorial, completará los siguientes pasos:

1. Cree un directorio del conjunto de pruebas
2. Crear archivos de configuración JSON
3. Cree el ejecutable del caso de prueba
4. Ejecute el conjunto de pruebas

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Requisitos del equipo host
  - Última versión de Device Tester AWS IoT
  - Python 3.7 o posterior

    Para comprobar la versión de Python instalada en su ordenador, ejecute el siguiente comando:

    ```
    python3 --version
    ```

    En Windows, si el uso de este comando devuelve un error, utilice `python --version` en su lugar. Si el número de versión devuelto es 3.7 o superior, ejecute el siguiente comando en una terminal de Powershell para configurar `python3` como alias para el comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Si no devuelve información sobre la versión o si la versión es inferior a 3.7, siga las instrucciones en Descargar Python para instalar Python 3.7+. Para obtener más información, consulte la documentación de Python.
  - urllib3

    Para comprobar que `urllib3` se ha instalado correctamente, ejecute el siguiente comando:

    ```
    python3 -c 'import urllib3'
    ```

    Si `urllib3` no está instalado, ejecute el siguiente comando para instalarlo:

    ```
    python3 -m pip install urllib3
    ```

- Requisitos de los dispositivos
  - Un dispositivo con un sistema operativo Linux y una conexión de red a la misma red que el equipo host.

    Le recomendamos que utilice un Raspberry Pi con el sistema operativo Raspberry Pi. Debe configurar SSH en su Raspberry Pi para conectarse de forma remota.

Cree un directorio del conjunto de pruebas

IDT separa de forma lógica los casos de prueba en grupos de pruebas dentro de cada conjunto de pruebas. Cada caso de prueba debe estar dentro de un grupo de prueba. Para este tutorial, cree una carpeta llamada `MyTestSuite_1.0.0` y cree el siguiente árbol de directorios dentro de esta carpeta:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Crear archivos de configuración JSON

El conjunto de pruebas debe contener los siguientes archivos de configuración JSON necesarios:

Archivos JSON necesarios

`suite.json`

Contiene información sobre el conjunto de pruebas. Consulte Configure suite.json.

`group.json`

Contiene información sobre un grupo de pruebas. Debe crear un archivo `group.json` para cada grupo de pruebas de su conjunto de pruebas. Consulte Configure group.json.

`test.json`

Contiene información sobre un caso de pruebas. Debe crear un archivo `test.json` para cada caso de pruebas de su conjunto de pruebas. Consulte Configure test.json.

1. En la carpeta `MyTestSuite_1.0.0/suite`, cree un archivo `suite.json` con la siguiente estructura:

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. En la carpeta `MyTestSuite_1.0.0/myTestGroup`, cree un archivo `group.json` con la siguiente estructura:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. En la carpeta `MyTestSuite_1.0.0/myTestGroup/myTestCase`, cree un archivo `test.json` con la siguiente estructura:

```
{
    "id": "MyTestCase",
    "title": "My Test Case",
    "details": "This is my test case.",
    "execution": {
        "timeout": 300000,
        "linux": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "mac": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "win": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
```

```
            ]
        }
    }
}
```

El árbol del directorio de su carpeta `MyTestSuite_1.0.0` ahora debe ser similar al siguiente:

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Obtenga el SDK de cliente de IDT

Utilce el SDK de cliente de IDT para permitir que IDT interactúe con el dispositivo que se está probando e informe de los resultados de las pruebas. Para este tutorial, utilizará la versión Python del SDK.

Desde la carpeta *<device-tester-extract-location>*`/sdks/python/`, copie la carpeta `idt_client` a su carpeta `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`.

Para comprobar que el SDK se ha copiado correctamente, ejecute el siguiente comando.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Cree el ejecutable del caso de prueba

Los ejecutables del caso de prueba contienen la lógica de prueba que desea ejecutar. Un conjunto de pruebas puede contener varios ejecutables de casos de prueba. Para este tutorial, creará solo un ejecutable de caso de prueba.

1. Cree el archivo del conjunto de pruebas.

   En la carpeta `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`, cree un archivo `myTestCase.py` con el siguiente contenido:

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

if __name__ == "__main__":
    main()
```

2. Utilice las funciones del SDK del cliente para añadir la siguiente lógica de prueba al archivo `myTestCase.py`:

   a. Ejecute un comando SSH en el dispositivo que se está probando.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

if __name__ == "__main__":
    main()
```

   b. Envíe el resultado de la prueba a IDT.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
```

```
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))

    # Send the result
    client.send_result(sr_req)

if __name__ == "__main__":
    main()
```

Configure la información del dispositivo para IDT

Configure la información del dispositivo para que IDT ejecute la prueba. Debe actualizar la plantilla `device.json` ubicada en la carpeta *<device-tester-extract-location>*`/configs` con la siguiente información.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
```

```
          }
        }
      }
    ]
  }
]
```

En el objeto `devices`, proporcione la siguiente información:

`id`

Un identificador único definido por el usuario para su dispositivo.

`connectivity.ip`

La dirección IP de su dispositivo.

`connectivity.port`

Opcional. El número de puerto que se va a utilizar para las conexiones SSH al dispositivo.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.user`

El nombre de usuario utilizado para iniciar sesión en el dispositivo.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada utilizada para iniciar sesión en su dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`devices.connectivity.auth.credentials.password`

La contraseña utilizada para iniciar sesión en su dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

> **Note**
>
> Especifique `privKeyPath` solo si `method` está establecido en `pki`
> Especifique `password` solo si `method` está establecido en `password`

Ejecute el conjunto de pruebas

Después de crear el conjunto de pruebas, querrá asegurarse de que funciona como se espera. Complete los siguientes pasos para ejecutar el conjunto de pruebas con su grupo de dispositivos existente para ello.

1. Copie su carpeta`MyTestSuite_1.0.0` en *<device-tester-extract-location>*`/tests`.
2. Ejecute los comandos siguientes:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT ejecuta su conjunto de pruebas y transmite los resultados a la consola. Cuando la prueba termine de ejecutarse, verá la siguiente información:

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=
```

```
========== Test Summary ==========
Execution Time:         1s
Tests Completed:        1
Tests Passed:           1
Tests Failed:           0
Tests Skipped:          0
---------------------------------
Test Groups:
    myTestGroup:        PASSED
---------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

Solución de problemas

Utilice la siguiente información para resolver cualquier problema que pueda surgir al completar el tutorial.

El caso de prueba no se ejecuta correctamente

Si la prueba no se ejecuta correctamente, IDT transmite los registros de errores a la consola para ayudarle a solucionar los problemas de la prueba. Antes de comprobar los registros de errores, verifique lo siguiente:

- El SDK de cliente de IDT se encuentra en la carpeta correcta, tal y como se describe en este paso.
- Cumpla con todos los requisitos previos de este tutorial.

No se puede conectar al dispositivo que se está probando

Compruebe lo siguiente:

- El archivo `device.json` contiene la dirección IP, el puerto y la información de autenticación correctos.
- Puede conectarse a su dispositivo a través de SSH desde su equipo host.

Cree archivos de configuración IDT para su conjunto de pruebas.

En esta sección se describen los formatos en los que se crean los archivos de configuración JSON que se incluyen al escribir un conjunto de pruebas personalizado.

Archivos JSON necesarios

`suite.json`

Contiene información sobre el conjunto de pruebas. Consulte Configure suite.json.

`group.json`

Contiene información sobre un grupo de pruebas. Debe crear un archivo `group.json` para cada grupo de pruebas de su conjunto de pruebas. Consulte Configure group.json.

`test.json`

Contiene información sobre un caso de pruebas. Debe crear un archivo `test.json` para cada caso de pruebas de su conjunto de pruebas. Consulte Configure test.json.

Archivos JSON opcionales

`state_machine.json`

Define cómo se ejecutan las pruebas cuando IDT ejecuta el conjunto de pruebas. Consulte Configure state_machine.json.

`userdata_schema.json`

Define el esquema del archivo`userdata.json` que los ejecutores de pruebas pueden incluir en su configuración de ajustes. El archivo `userdata.json` se utiliza para cualquier información de configuración adicional necesaria para ejecutar la prueba, pero que no esté presente en el archivo `device.json`. Consulte Configura userdata_schema.json.

Los archivos de configuración JSON se colocan en su archivo *<custom-test-suite-folder>*, tal y como se muestra aquí.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### state_machine.json
```

```
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configure suite.json

El archivo `suite.json` establece las variables de entorno y determina si los datos del usuario son necesarios para ejecutar el conjunto de pruebas. Utilice la siguiente plantilla para configurar el *<custom-test-suite-folder>*`/suite/suite.json` archivo:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un identificador único definido por el usuario para el conjunto de pruebas. El valor de `id` debe coincidir con el nombre de la carpeta del conjunto de pruebas en la que se encuentra el archivo `suite.json`. El nombre y la versión del paquete también deben cumplir los siguientes requisitos:

- *<suite-name>* puede contener guiones bajos.
- *<suite-version>* se denota como *x.x.x*, donde x es un número.

El identificador se muestra en los informes de prueba generados por IDT.

`title`

Un nombre definido por el usuario para el producto o la característica que se está probando en este conjunto de pruebas. El nombre se muestra en la CLI de IDT para los ejecutores de pruebas.

`details`

Una breve descripción de la finalidad del conjunto de pruebas.

`userDataRequired`

Define si los ejecutores de pruebas deben incluir información personalizada en un archivo `userdata.json`. Si establece este valor en `true`, también debe incluir el archivo `userdata_schema.json` en la carpeta del conjunto de pruebas.

`environmentVariables`

Opcional. Una matriz de variables de entorno que se va a configurar para este conjunto de pruebas.

`environmentVariables.key`

El nombre de la variable de entorno.

`environmentVariables.value`

El valor de la variable de entorno.

Configure group.json

El archivo `group.json` define si el grupo de prueba es obligatorio u opcional. Utilice la siguiente plantilla para configurar el *<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json` archivo:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un identificador único definido por el usuario para el conjunto de pruebas. El valor de `id` debe coincidir con el nombre de la carpeta del grupo de pruebas en la que se encuentra el archivo `group.json` y no debe contener guiones bajos (_). El identificador se utiliza en los informes de prueba generados por IDT.

`title`

Nombre descriptivo para el grupo de prueba. El nombre se muestra en la CLI de IDT para los ejecutores de pruebas.

`details`

Una breve descripción de la finalidad del grupo de pruebas.

`optional`

Opcional. Configure `true` para mostrar este grupo de pruebas como un grupo opcional una vez que IDT termine de ejecutar las pruebas requeridas. El valor predeterminado es `false`.

Configure test.json

El archivo `test.json` determina los ejecutables del caso de prueba y las variables de entorno que utiliza un caso de prueba. Para obtener más información acerca de la creación de ejecutables de casos de prueba, consulte Cree ejecutables de casos de prueba de IDT.

Utilice la siguiente plantilla para configurar el *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json archivo:

```
{
    "id": "<test-id>",
    "title": "<test-title>",
    "details": "<test-details>",
    "requireDUT": true | false,
    "requiredResources": [
        {
            "name": "<resource-name>",
            "features": [
                {
                    "name": "<feature-name>",
                    "version": "<feature-version>",
```

```
                    "jobSlots": <job-slots>
                }
            ]
        }
    ],
    "execution": {
        "timeout": <timeout>,
        "mac": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "linux": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "win": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ]
        }
    },
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un identificador único definido por el usuario para el caso de pruebas. El valor de `id` debe coincidir con el nombre de la carpeta del caso de pruebas en la que se encuentra el archivo `test.json` y no debe contener guiones bajos (_). El identificador se utiliza en los informes de prueba generados por IDT.

`title`

Nombre descriptivo para el caso de prueba. El nombre se muestra en la CLI de IDT para los ejecutores de pruebas.

`details`

Una breve descripción de la finalidad del caso de pruebas.

`requireDUT`

Opcional. Configure en `true` si se requiere un dispositivo para ejecutar esta prueba; de lo contrario, ajústelo en `false`. El valor predeterminado es `true`. Los ejecutores de la prueba configurarán los dispositivos que utilizarán para ejecutar la prueba en su archivo `device.json`.

`requiredResources`

Opcional. Una matriz que proporciona información sobre los dispositivos de recursos necesarios para ejecutar esta prueba.

`requiredResources.name`

El nombre exclusivo que se le dará al dispositivo de recursos cuando se ejecute esta prueba.

`requiredResources.features`

Conjunto de características del dispositivo de recursos definidas por el usuario.

`requiredResources.features.name`

El nombre de la característica. La característica del dispositivo para la que desea utilizar este dispositivo. Este nombre coincide con el nombre de la característica proporcionado por el ejecutor de pruebas en el archivo `resource.json`.

`requiredResources.features.version`

Opcional. La versión de la característica. Este valor se compara con la versión de la característica proporcionada por el ejecutor de pruebas en el archivo `resource.json`. Si no se proporciona una versión, la característica no está marcada. Si el número de versión no es necesario para la característica, deje este campo en blanco.

`requiredResources.features.jobSlots`

Opcional. El número de pruebas simultáneas que puede admitir esta característica. El valor predeterminado es 1. Si desea que IDT utilice distintos dispositivos para características individuales, le recomendamos que establezca este valor en 1.

`execution.timeout`

La cantidad de tiempo (en milisegundos) que IDT espera a que la prueba termine de ejecutarse. Para obtener más información sobre cómo fijar este valor, consulte Cree ejecutables de casos de prueba de IDT.

`execution.os`

Los ejecutables del caso de prueba que se ejecutarán en función del sistema operativo del equipo host que ejecuta IDT. Los valores admitidos son `linux`, `mac` y `win`.

`execution.os.cmd`

La ruta al ejecutable del caso de prueba que desea ejecutar para el sistema operativo especificado. Esta ubicación debe estar en la ruta del sistema.

`execution.os.args`

Opcional. Los argumentos que se deben proporcionar para ejecutar el ejecutable del caso de prueba.

`environmentVariables`

Opcional. Una matriz de variables de entorno que se va a configurar para este caso de pruebas.

`environmentVariables.key`

El nombre de la variable de entorno.

`environmentVariables.value`

El valor de la variable de entorno.

> **Note**
>
> Si especifica la misma variable de entorno en el archivo `test.json` y en el archivo `suite.json`, el valor del archivo `test.json` tiene prioridad.

Configure state_machine.json

Una máquina de estados es un constructo que controla el flujo de ejecución del conjunto de pruebas. Determina el estado inicial de un conjunto de pruebas, gestiona las transiciones de estado en función de las reglas definidas por el usuario y continúa pasando por esos estados hasta alcanzar el estado final.

Si su conjunto de pruebas no incluye una máquina de estados definida por el usuario, IDT generará una máquina de estados para usted. La máquina de estados predeterminada realiza las siguientes funciones:

- Ofrece a los ejecutores de pruebas la posibilidad de seleccionar y ejecutar grupos de pruebas específicos, en lugar de todo el conjunto de pruebas.
- Si no se seleccionan grupos de pruebas específicos, ejecuta todos los grupos de pruebas del conjunto de pruebas con asignación al azar.
- Genera informes e imprime un resumen de la consola que muestra los resultados de las pruebas de cada grupo y caso de prueba.

Para obtener más información sobre cómo funciona la máquina de estados IDT, consulte Configure la máquina de estados IDT.

Configura userdata_schema.json

El archivo `userdata_schema.json` determina el esquema en el que los ejecutores de pruebas proporcionan los datos de los usuarios. Los datos de usuario son necesarios si su conjunto de pruebas requiere información que no está presente en el archivo `device.json`. Por ejemplo, es posible que las pruebas necesiten credenciales de red Wi-Fi, puertos abiertos específicos o certificados que deba proporcionar un usuario. Esta información se puede proporcionar a IDT como un parámetro de entrada denominado `userdata`, cuyo valor es un archivo `userdata.json`, que los usuarios crean en su carpeta *`<device-tester-extract-location>`*`/config`. El formato del archivo `userdata.json` se basa en el archivo `userdata_schema.json` que incluya en el conjunto de pruebas.

Para indicar que los ejecutores de pruebas deben proporcionar un archivo `userdata.json`:

1. En el archivo `suite.json`, ajuste `userDataRequired` a `true`.
2. En su *`<custom-test-suite-folder>`*, cree un archivo `userdata_schema.json`.
3. Edite el archivo `userdata_schema.json` para crear un borrador del esquema JSON v4 del IETF válido.

Cuando IDT ejecuta su conjunto de pruebas, lee automáticamente el esquema y lo usa para validar el archivo `userdata.json` proporcionado por el ejecutor de la prueba. Si es válido, el contenido del archivo `userdata.json` está disponible tanto en el contexto IDT como en el contexto de la máquina de estados.

Configure la máquina de estados IDT

Una máquina de estados es un constructo que controla el flujo de ejecución del conjunto de pruebas. Determina el estado inicial de un conjunto de pruebas, gestiona las transiciones de estado en función de las reglas definidas por el usuario y continúa pasando por esos estados hasta alcanzar el estado final.

Si su conjunto de pruebas no incluye una máquina de estados definida por el usuario, IDT generará una máquina de estados para usted. La máquina de estados predeterminada realiza las siguientes funciones:

- Ofrece a los ejecutores de pruebas la posibilidad de seleccionar y ejecutar grupos de pruebas específicos, en lugar de toda el conjunto de pruebas.
- Si no se seleccionan grupos de pruebas específicos, ejecuta todos los grupos de pruebas del conjunto de pruebas en orden aleatorio.
- Genera informes e imprime un resumen de la consola que muestra los resultados de las pruebas de cada grupo y caso de prueba.

La máquina de estado de un conjunto de pruebas de IDT debe cumplir los siguientes criterios:

- Cada estado corresponde a una acción que debe realizar IDT, como ejecutar un grupo de pruebas o crear un archivo de informe.
- La transición a un estado ejecuta la acción asociada al estado.
- Cada estado define la regla de transición para el estado siguiente.
- El estado final debe ser `Succeed` o bien `Fail`.

Formato de las máquinas de estado

Puede utilizar la siguiente plantilla para configurar su propio archivo *<custom-test-suite-folder>*`/suite/state_machine.json`:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
```

```
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Comment`

Una descripción de la máquina de estado.

`StartAt`

El nombre del estado en el que IDT comienza a ejecutar el conjunto de pruebas. El valor de `StartAt` debe estar establecido en uno de los estados enumerados en el objeto `States`.

`States`

Un objeto que asigna los nombres de estado definidos por usuario a estados IDT válidos. Cada estado. El objeto de *`nombre de estado`* contiene la definición de un estado válido asignado al *`nombre del estado.`*

El objeto `States` debe incluir los estados `Succeed` y `Fail`. Para obtener información sobre los estados válidos, consulte Estados válidos y definiciones de estado.

Estados válidos y definiciones de estado

En esta sección se describen las definiciones de estado de todos los estados válidos que se pueden usar en la máquina de estados de IDT. Algunos de los siguientes estados admiten configuraciones a nivel de caso de prueba. Sin embargo, le recomendamos que configure las reglas de transición de estado a nivel de grupo de prueba en lugar de a nivel del caso de prueba, a menos que sea absolutamente necesario.

Definiciones de estado

- RunTask
- Opción
- Parallel
- AgregueCaracterísticasProducto
- Informar
- LogMessage
- SelectGroup
- Fail
- Succeed

RunTask

El estado `RunTask`ejecuta casos de prueba a partir de un grupo de pruebas definido en el conjunto de pruebas.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`TestGroup`

Opcional. El ID del grupo de pruebas que se va a ejecutar. Si no se especifica este valor, IDT ejecuta el grupo de pruebas que seleccione el ejecutor de la prueba.

`TestCases`

Opcional. Un conjunto de identificadores de casos de prueba del grupo especificado en `TestGroup`. En función de los valores de `TestGroup` y `TestCases`, IDT determina el comportamiento de la ejecución de la prueba de la siguiente manera:

- Cuando se especifican ambos `TestGroup` y `TestCases`, IDT ejecuta los casos de prueba especificados del grupo de pruebas.
- Cuando `TestCases` se especifican pero no se especifica `TestGroup`, IDT ejecuta los casos de prueba especificados.
- Cuando `TestGroup` se especifica, pero no se especifica `TestCases`, IDT ejecuta todos los casos de prueba del grupo de pruebas especificado.
- Si no se especifica ninguno, `TestGroup` o `TestCases`, IDT ejecuta todos los casos de prueba del grupo de pruebas que el ejecutor de la prueba selecciona en la CLI de IDT. Para habilitar la selección de grupos para los participantes en las pruebas, debe incluir ambos estados `RunTask` y `Choice` en el archivo `statemachine.json`. Para ver un ejemplo de cómo funciona, consulte Ejemplo de máquina de estados: ejecutar grupos de prueba seleccionados por el usuario.

  Para obtener más información sobre cómo habilitar los comandos CLI de IDT para los ejecutores de pruebas, consulte the section called “Habilitar comandos CLI de IDT”.

`ResultVar`

El nombre de la variable de contexto que se va a configurar con los resultados de la prueba. No especifique este valor si no especificó ningún valor para `TestGroup`. IDT establece el valor de la variable que usted define en `ResultVar` para `true` o `false` en función de lo siguiente:

- Si el nombre de la variable tiene el formato *`text`*`_`*`text`*`_passed`, el valor se establece en función de si todas las pruebas del primer grupo de pruebas se aprobaron o se omitieron.
- En todos los demás casos, el valor se establece en función de si todas las pruebas de todos los grupos de pruebas se aprobaron o se omitieron.

Normalmente, utilizará el estado `RunTask` para especificar un identificador de grupo de pruebas sin especificar los identificadores de casos de prueba individuales, de modo que IDT ejecutará todos los casos de prueba del grupo de pruebas especificado. Todos los casos de prueba ejecutados por este estado se ejecutan en paralelo, en orden aleatorio. Sin embargo, si todos los casos de prueba requieren la ejecución de un dispositivo y solo hay un dispositivo disponible, los casos de prueba se ejecutarán secuencialmente.

Error handling (Control de errores)

Si alguno de los grupos de pruebas especificados o identificadores de casos de prueba no es válido, este estado genera el error de ejecución `RunTaskError`. Si el estado encuentra un error de ejecución, también establece la variable `hasExecutionError` en el contexto de la máquina de estados en `true`.

Opción

El estado `Choice` le permite configurar dinámicamente el siguiente estado al que realizar la transición en función de las condiciones definidas por el usuario.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Default`

El estado predeterminado al que se realizará la transición si no se puede evaluar ninguna de las expresiones definidas en `Choices` para `true`.

`FallthroughOnError`

Opcional. Especifica el comportamiento cuando el estado encuentra un error al evaluar las expresiones. Configúrelo en `true` si desea omitir una expresión si la evaluación produce un error. Si ninguna expresión coincide, la máquina de estados pasa al estado `Default`. Si no se especifica, el valor de `FallthroughOnError` se establece de forma predeterminada en `false`.

`Choices`

Una matriz de expresiones y estados para determinar a qué estado hacer la transición después de ejecutar las acciones en el estado actual.

`Choices.Expression`

Una expresión que debe evaluarse como un valor booleano. Si la expresión se evalúa como `true`, la máquina de estados pasa al estado definido en `Choices.Next`. Las cadenas de expresión recuperan los valores del contexto de la máquina de estados y, a continuación, realizan operaciones en ellos para obtener un valor booleano. Para obtener información sobre cómo acceder al contexto de la máquina de estados, consulte Contexto de las máquinas de estados.

`Choices.Next`

El nombre del estado al que se realizará la transición si la expresión definida en `Choices.Expression` se evalúa como `true`.

Error handling (Control de errores)

El estado `Choice` puede requerir la gestión de errores en los siguientes casos:

- Algunas variables de las expresiones de elección no existen en el contexto de la máquina de estados.
- El resultado de una expresión no es un valor booleano.
- El resultado de una búsqueda en JSON no es una cadena, un número ni un booleano.

No puede usar un bloque `Catch` para gestionar los errores en este estado. Si quiere detener la ejecución de la máquina de estados cuando encuentre un error, debe configurar `FallthroughOnError` en `false`. Sin embargo, le recomendamos que configure `FallthroughOnError` en `true` y, en función de su caso de uso, haga una de las siguientes opciones:

- Si se espera que una variable a la que está accediendo no exista en algunos casos, utilice el valor `Default` y los bloques `Choices` adicionales para especificar el siguiente estado.
- Si una variable a la que está accediendo debe existir siempre, defina el estado `Default` en `Fail`.

Parallel

El estado `Parallel` le permite definir y ejecutar nuevas máquinas de estados en paralelo entre sí.

```
{
```

```
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`Branches`

Un conjunto de definiciones de máquinas de estados para ejecutar. Cada definición de máquina de estados debe contener su propio `StartAt`, `Succeed` y estados `Fail`. Las definiciones de máquinas de estados de esta matriz no pueden hacer referencia a estados ajenos a su propia definición.

> **Note**
>
> Como cada máquina de estado de rama comparte el mismo contexto de máquina de estados, establecer variables en una rama y, a continuación, leer esas variables de otra rama podría provocar un comportamiento inesperado.

El estado `Parallel` pasa al siguiente estado solo después de ejecutar todas las máquinas de estado de rama. Cada estado que requiera un dispositivo esperará para ejecutarse hasta que el dispositivo esté disponible. Si hay varios dispositivos disponibles, este estado ejecuta casos de prueba de varios grupos en paralelo. Si no hay suficientes dispositivos disponibles, los casos de prueba se ejecutarán secuencialmente. Como los casos de prueba se ejecutan en orden aleatorio cuando se ejecutan en paralelo, se podrían usar diferentes dispositivos para ejecutar pruebas del mismo grupo de pruebas.

Error handling (Control de errores)

Asegúrese de que tanto la máquina de estado de la rama como la máquina de estados principal pasen al estado `Fail` para gestionar los errores de ejecución.

Como las máquinas de estado de rama no transmiten los errores de ejecución a la máquina de estado principal, no puede usar un bloque `Catch` para gestionar los errores de ejecución en las máquinas de estado de rama. En su lugar, utilice el valor `hasExecutionErrors` en el contexto de la máquina de estado compartida. Para ver un ejemplo de cómo funciona, consulte Ejemplo de máquina de estados: ejecute dos grupos de prueba en paralelo.

AgregueCaracterísticasProducto

El estado `AddProductFeatures` le permite añadir características del producto al archivo `awsiotdevicetester_report.xml` generado por IDT.

Una característica del producto es información definida por el usuario sobre los criterios específicos que puede cumplir un dispositivo. Por ejemplo, la característica `MQTT` del producto puede indicar que el dispositivo publica los mensajes MQTT correctamente. En el informe, las características del producto se establecen como `supported`, `not-supported`, o como un valor personalizado, en función de si se han superado las pruebas especificadas.

> **Note**
>
> El estado `AddProductFeatures` no genera informes por sí mismo. Este estado debe realizar la transición al estado `Report` para generar informes.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
```

```
                "<execution-method>"
            ]
        }
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`Features`

Un conjunto de características del producto para mostrar en el archivo `awsiotdevicetester_report.xml`.

`Feature`

El nombre de la característica

`FeatureValue`

Opcional. El valor personalizado que se utilizará en el informe en lugar de `supported`. Si no se especifica este valor, según los resultados de las pruebas, el valor de la característica se establece en `supported` o `not-supported`.

Si utiliza un valor personalizado para `FeatureValue`, puede probar la misma característica con diferentes condiciones e IDT concatena los valores de la característica para las condiciones admitidas. Por ejemplo, en el siguiente fragmento se muestra la característica `MyFeature` con dos valores de característica distintos:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
```

```
},
...
```

Si ambos grupos de pruebas aprueban, el valor de la característica se establece en `first-feature-supported, second-feature-supported`.

`Groups`

Opcional. Una matriz de los ID de los grupos de prueba. Para que la característica sea compatible, deben aprobarse todas las pruebas de cada grupo de pruebas especificado.

`OneOfGroups`

Opcional. Una matriz de los ID de los grupos de prueba. Para que la característica sea compatible, deben aprobarse todas las pruebas de al menos uno de los grupos de pruebas especificados.

`TestCases`

Opcional. Una matriz de los ID de los casos de prueba. Si especifica este valor, se aplicará lo siguiente:

- Se deben aprobar todos los casos de prueba especificados para que la característica sea compatible.
- `Groups` debe contener solo un ID de grupo de prueba.
- `OneOfGroups` no debe especificarse.

`IsRequired`

Opcional. Configúrelo en `false` para marcar esta característica como una característica opcional en el informe. El valor predeterminado es `true`.

`ExecutionMethods`

Opcional. Un conjunto de métodos de ejecución que coinciden con el valor `protocol` especificado en el archivo `device.json`. Si se especifica este valor, los ejecutores de pruebas deben especificar un valor `protocol` que coincida con uno de los valores de esta matriz para incluir la característica en el informe. Si no se especifica este valor, la característica siempre se incluirá en el informe.

Para usar el estado `AddProductFeatures`, debe establecer el valor de `ResultVar`del estado `RunTask` en uno de los siguientes valores:

- Si especificó los ID de los casos de prueba individuales, configure `ResultVar` en *`group-id_test-id`*`_passed`.
- Si no especificó los ID de los casos de prueba individuales, configure `ResultVar` en *`group-id`*`_passed`.

El estado `AddProductFeatures` comprueba los resultados de las pruebas de la siguiente manera:

- Si no especificó ningún identificador de caso de prueba, el resultado de cada grupo de prueba se determina a partir del valor de la variable *`group-id`*`_passed` en el contexto de la máquina de estados.
- Si especificó los ID de los casos de prueba, el resultado de cada una de las pruebas se determina a partir del valor de la variable *`group-id_test-id`*`_passed` en el contexto de la máquina de estados.

Error handling (Control de errores)

Si un identificador de grupo proporcionado en este estado no es un identificador de grupo válido, este estado provoca un error `AddProductFeaturesError` de ejecución. Si el estado encuentra un error de ejecución, también establece la variable `hasExecutionErrors` en el contexto de la máquina de estados en `true`.

Informar

El estado `Report` genera los archivos *`suite-name`*`_Report.xml` y `awsiotdevicetester_report.xml`. Este estado también transmite el informe a la consola.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

Siempre debe pasar al estado `Report` que se encuentra al final del flujo de ejecución de la prueba para que los ejecutores de la prueba puedan ver los resultados de la prueba. Normalmente, el siguiente estado después de este estado es `Succeed`.

Error handling (Control de errores)

Si este estado tiene problemas con la generación de los informes, se produce el error de ejecución `ReportError`.

LogMessage

El estado `LogMessage` genera el archivo `test_manager.log` y transmite el mensaje de registro a la consola.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`Level`

El nivel de error en el que se va a crear el mensaje de registro. Si especifica un nivel que no es válido, este estado genera un mensaje de error y lo descarta.

`Message`

El mensaje para registrar.

SelectGroup

El estado `SelectGroup` actualiza el contexto de la máquina de estados para indicar qué grupos están seleccionados. Los valores establecidos por este estado se utilizan en cualquier estado `Choice` posterior.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`TestGroups`

Una matriz de grupos de prueba que se marcarán como seleccionados. Para cada ID de grupo de prueba de esta matriz, la variable *`group-id`*`_selected` se establece `true` en el contexto. Asegúrese de proporcionar identificadores de grupos de prueba válidos, ya que el IDT no valida que existen los grupos especificados.

Fail

El estado `Fail` indica que la máquina de estados no se ejecutó correctamente. Este es el estado final de la máquina de estados y cada definición de la máquina de estados debe incluir este estado.

```
{
    "Type": "Fail"
}
```

Succeed

El estado `Succeed` indica que la máquina de estados se ejecutó correctamente. Este es el estado final de la máquina de estados y cada definición de la máquina de estados debe incluir este estado.

```
{
    "Type": "Succeed"
}
```

Contexto de las máquinas de estados

El contexto de la máquina de estados es un documento JSON de solo lectura que contiene datos que están disponibles para la máquina de estados durante la ejecución. Solo se puede acceder al contexto de la máquina de estados desde la máquina de estados y contiene información que determina el flujo de prueba. Por ejemplo, puede usar la información configurada por los ejecutores de la prueba en el archivo `userdata.json` para determinar si es necesaria la ejecución de una prueba específica.

El contexto de la máquina de estados usa el siguiente formato:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    },
    "suiteFailed": true | false,
    "specificTestGroups": [
        "<group-id>"
    ],
    "specificTestCases": [
        "<test-id>"
    ],
    "hasExecutionErrors": true
}
```

`pool`

Información sobre el grupo de dispositivos seleccionado para la ejecución de la prueba. Para un grupo de dispositivos seleccionados, esta información se recupera del elemento correspondiente de la matriz del grupo de dispositivos de alto nivel definido en el archivo `device.json`.

`userData`

Información en el archivo `userdata.json`

`config`

La información se sujeta en el archivo `config.json`.

`suiteFailed`

El valor se establece en `false` cuando se inicia la máquina de estados. Si un grupo de pruebas falla en un estado `RunTask`, este valor se establece en `true` durante el resto de la ejecución de la máquina de estados.

`specificTestGroups`

Si el ejecutor de la prueba selecciona grupos de pruebas específicos para ejecutarlos en lugar de todo el conjunto de pruebas, se crea esta clave y contiene la lista de ID de grupos de pruebas específicos.

`specificTestCases`

Si el ejecutor de la prueba selecciona casos de pruebas específicos para ejecutarlos en lugar de todo el conjunto de pruebas, se crea esta clave y contiene la lista de ID de casos de pruebas específicos.

`hasExecutionErrors`

No se cierra cuando se inicia la máquina de estados. Si algún estado detecta un error de ejecución, se crea esta variable y se establece en `true` durante el resto de la ejecución de la máquina de estados.

Puede consultar el contexto mediante la notación JSONPath. La sintaxis de las consultas de JSONPath en las definiciones de estado es `{{$.query}}`. Puede utilizar las consultas de JSONPath como cadenas de marcadores de posición en algunos estados. IDT reemplaza las cadenas de marcadores de posición con el valor de la consulta JSONPath evaluada del contexto. Puede utilizar los siguientes marcadores de posición para los siguientes valores

- El valor `TestCases` en estados `RunTask`.
- El valor `Expression` estado `Choice`.

Cuando accede a los datos del contexto de las máquinas de estado, asegúrese de que se cumplan las siguientes condiciones:

- Sus rutas de JSON deben comenzar por `$.`

- Cada valor debe dar como resultado una cadena, un número o un booleano.

Para obtener más información sobre el uso de la notación JSONPath para acceder a los datos del contexto, consulte Utilice el contexto de IDT.

Errores de ejecución

Los errores de ejecución son errores en la definición de la máquina de estados que la máquina de estados encuentra al ejecutar un estado. IDT registra la información sobre cada error en el archivo `test_manager.log` y transmite el mensaje de registro a la consola.

Puede utilizar los siguientes métodos para gestionar los errores de ejecución:

- Añada un bloque `Catch` a la definición de estado.
- Compruebe el valor del valor `hasExecutionErrors` en el contexto de la máquina de estados.

Captura

Para usar `Catch`, añada lo siguiente a su definición de estado:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Catch.ErrorEquals`

> Una matriz de los tipos de error que se deben capturar. Si un error de ejecución coincide con uno de los valores especificados, la máquina de estados pasa al estado especificado en `Catch.Next`. Consulte la definición de cada estado para obtener información sobre el tipo de error que produce.

`Catch.Next`

El siguiente estado al que se realizará la transición si el estado actual encuentra un error de ejecución que coincide con uno de los valores especificados en `Catch.ErrorEquals`.

Los bloques de captura se gestionan secuencialmente hasta que uno coincide. Si los errores no coinciden con los enumerados en los bloques de Captura, las máquinas de estado seguirán ejecutándose. Como los errores de ejecución son el resultado de definiciones de estado incorrectas, se recomienda que pase al estado de Falla cuando un estado detecte un error de ejecución.

hasExecutionError

Cuando algunos estados encuentran errores de ejecución, además de emitir el error, también establecen el valor `hasExecutionError` en `true` en el contexto de la máquina de estados. Puede usar este valor para detectar cuando se produce un error y, a continuación, use un estado `Choice` para hacer la transición de la máquina de estados al estado `Fail`.

Este método incluye las siguientes características:

- La máquina de estados no comienza con ningún valor asignado a `hasExecutionError` y este valor no está disponible hasta que se establezca en un estado concreto. Esto significa que debe establecer explícitamente el valor `FallthroughOnError` en `false` para los estados `Choice` que acceden a este valor para evitar que la máquina de estados se detenga si no se produce ningún error de ejecución.
- Una vez establecido en `true`, `hasExecutionError` nunca se establece en falso ni se elimina del contexto. Esto significa que este valor solo es útil la primera vez que se establece en `true`, y para todos los estados posteriores, no proporciona un valor significativo.
- El valor `hasExecutionError` se comparte con todas las máquinas de estado ramificada en el estado `Parallel`, lo que puede provocar resultados inesperados en función del orden en que se acceda a él.

Debido a estas características, no recomendamos utilizar este método si se puede utilizar un bloque de Captura en su lugar.

Máquinas de estado de ejemplo

En esta sección se proporcionan algunos ejemplos de configuraciones de máquinas de estado.

Ejemplos

- Ejemplo de máquina de estados: ejecute un solo grupo de pruebas
- Ejemplo de máquina de estados: ejecute grupos de prueba seleccionados por el usuario
- Ejemplo de máquina de estados: ejecute un solo grupo de pruebas con características de productos
- Ejemplo de máquina de estados: ejecute dos grupos de prueba en paralelo

Ejemplo de máquina de estados: ejecute un solo grupo de pruebas

Esta máquina de estado:

- Ejecuta el grupo de prueba con una identificación `GroupA`, que debe estar presente en la suite en un archivo `group.json`.
- Comprueba si hay errores de ejecución y realiza transiciones a `Fail` si se encuentra alguno.
- Genera un informe y hace la transición a `Succeed` si no hay errores y `Fail` de todas formas.

```
{
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
```

```
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Ejemplo de máquina de estados: ejecute grupos de prueba seleccionados por el usuario

Esta máquina de estado:

- Comprueba si el ejecutor de pruebas seleccionó grupos de pruebas específicos. La máquina de estados no comprueba si hay casos de prueba específicos porque los ejecutores de pruebas no pueden seleccionar casos de prueba sin seleccionar también un grupo de pruebas.
- Si se seleccionan grupos de prueba:
  - Ejecuta los casos de prueba dentro de los grupos de prueba seleccionados. Para ello, la máquina de estados no especifica explícitamente ningún grupo de pruebas o casos de prueba en el estado `RunTask`.
  - Genera un informe después de ejecutar todas las pruebas y salidas.
- Si no se seleccionan grupos de prueba:
  - Ejecuta las pruebas en el grupo de pruebas `GroupA`.
  - Genera informes y salidas.

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise
 runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
```

```
            "Choices": [
                {
                    "Expression": "{{$.specificTestGroups[0]}} != ''",
                    "Next": "RunSpecificGroups"
                }
            ]
        },
        "RunSpecificGroups": {
            "Type": "RunTask",
            "Next": "Report",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
```

```
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Ejemplo de máquina de estados: ejecute un solo grupo de pruebas con características de productos

Esta máquina de estado:

- Ejecuta el grupo de prueba `GroupA`.
- Comprueba si hay errores de ejecución y realiza transiciones a `Fail` si se encuentra alguno.
- Añade la característica `FeatureThatDependsOnGroupA` al archivo `awsiotdevicetester_report.xml`:
  - Si se aprueba `GroupA`, la característica se establece en `supported`.
  - La característica no está marcada como opcional en el informe.
- Genera un informe y hace la transición a `Succeed` si no hay errores y `Fail` de todas formas.

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
```

```
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Ejemplo de máquina de estados: ejecute dos grupos de prueba en paralelo

Esta máquina de estado:

- Ejecuta los grupos de prueba `GroupA` y `GroupB` en paralelo. Las variables `ResultVar` almacenadas en el contexto por los estados `RunTask` de las máquinas de estado de ramificación están disponibles para el estado `AddProductFeatures`.

- Comprueba si hay errores de ejecución y realiza transiciones a `Fail` si se encuentra alguno. Esta máquina de estados no utiliza un bloque `Catch` porque ese método no detecta errores de ejecución en las máquinas de estados de ramificación.
- Agrega características al archivo `awsiotdevicetester_report.xml` en función de los grupos que pasan
  - Si se aprueba `GroupA`, la característica se establece en `supported`.
  - La característica no está marcada como opcional en el informe.
- Genera un informe y hace la transición a `Succeed` si no hay errores y `Fail` de todas formas.

Si hay dos dispositivos configurados en el grupo de dispositivos, ambos `GroupA` y `GroupB` pueden ejecutarse al mismo tiempo. Sin embargo, si `GroupA` o `GroupB` tienen varias incluyen varias pruebas, es posible que ambos dispositivos se asignen a esas pruebas. Si solo se configura un dispositivo, los grupos de prueba se ejecutarán secuencialmente.

```
{
    "Comment": "Runs GroupA and GroupB in parallel",
    "StartAt": "RunGroupAAndB",
    "States": {
        "RunGroupAAndB": {
            "Type": "Parallel",
            "Next": "CheckForErrors",
            "Branches": [
                {
                    "Comment": "Run GroupA state machine",
                    "StartAt": "RunGroupA",
                    "States": {
                        "RunGroupA": {
                            "Type": "RunTask",
                            "Next": "Succeed",
                            "TestGroup": "GroupA",
                            "ResultVar": "GroupA_passed",
                            "Catch": [
                                {
                                    "ErrorEquals": [
                                        "RunTaskError"
                                    ],
                                    "Next": "Fail"
                                }
                            ]
                        },
```

```
                    "Succeed": {
                        "Type": "Succeed"
                    },
                    "Fail": {
                        "Type": "Fail"
                    }
                }
            },
            {
                "Comment": "Run GroupB state machine",
                "StartAt": "RunGroupB",
                "States": {
                    "RunGroupA": {
                        "Type": "RunTask",
                        "Next": "Succeed",
                        "TestGroup": "GroupB",
                        "ResultVar": "GroupB_passed",
                        "Catch": [
                            {
                                "ErrorEquals": [
                                    "RunTaskError"
                                ],
                                "Next": "Fail"
                            }
                        ]
                    },
                    "Succeed": {
                        "Type": "Succeed"
                    },
                    "Fail": {
                        "Type": "Fail"
                    }
                }
            }
        ]
    },
    "CheckForErrors": {
        "Type": "Choice",
        "Default": "AddProductFeatures",
        "FallthroughOnError": true,
        "Choices": [
            {
                "Expression": "{{$.hasExecutionErrors}} == true",
                "Next": "Fail"
```

```
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                },
                {
                    "Feature": "FeatureThatDependsOnGroupB",
                    "Groups": [
                        "GroupB"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Cree ejecutables de casos de prueba de IDT

Puede crear y colocar ejecutables de casos de prueba en una carpeta de conjunto de pruebas de las siguientes maneras:

- En el caso de los conjuntos de pruebas que utilizan argumentos o variables de entorno de los archivos `test.json` para determinar qué pruebas se van a ejecutar, puede crear un único ejecutable de caso de prueba para todo el conjunto de pruebas o un ejecutable de prueba para cada grupo de pruebas del conjunto de pruebas.
- En el caso de un conjunto de pruebas en el que desee ejecutar pruebas específicas en función de comandos específicos, debe crear un caso de prueba ejecutable para cada caso de prueba del conjunto de pruebas.

Como redactor de pruebas, puede determinar qué enfoque es adecuado para su caso de uso y estructurar el ejecutable del caso de prueba en consecuencia. Asegúrese de proporcionar la ruta de acceso correcta al ejecutable del caso de prueba en cada archivo `test.json` y de que el ejecutable especificado se ejecute correctamente.

Cuando todos los dispositivos estén preparados para ejecutar un caso de prueba, IDT lee los siguientes archivos:

- El `test.json` para el caso de prueba seleccionado determina los procesos que se van a iniciar y las variables de entorno que se van a configurar.
- El `suite.json` para el conjunto de pruebas determina las variables de entorno que se van a configurar.

IDT inicia el proceso ejecutable de prueba requerido en función de los comandos y argumentos especificados en el archivo `test.json` y pasa las variables de entorno requeridas al proceso.

Utilice el SDK de cliente de IDT

Los SDK de cliente de IDT le permiten simplificar la forma de escribir la lógica de prueba en su ejecutable de prueba con comandos de API que puede utilizar para interactuar con IDT y los dispositivos que se están probando. En la actualidad, IDT ofrece los siguientes SDK:

- SDK de cliente de IDT para Python
- SDK de cliente de IDT para Go

Estos SDK se encuentran en la carpeta *<device-tester-extract-location>*/sdks. Al crear un ejecutable de caso de prueba nuevo, debe copiar el SDK que quiere usar en la carpeta que contiene el ejecutable del caso de prueba y hacer referencia al SDK en su código. En esta sección se proporciona una breve descripción de los comandos de API disponibles que puede usar en los ejecutables de casos de prueba.

En esta sección

- Interacción del dispositivo
- Interacción IDT
- Interacción con el host

Interacción del dispositivo

Los siguientes comandos le permiten comunicarse con el dispositivo que se está probando sin tener que implementar ninguna función adicional de administración de la conectividad e interacción del dispositivo.

`ExecuteOnDevice`

Permite que los conjuntos de pruebas ejecuten intérpretes de comandos en un dispositivo que admite conexiones de intérpretes de comandos SSH o Docker.

`CopyToDevice`

Permite a los conjuntos de pruebas copiar un archivo local desde la máquina host que ejecuta IDT a una ubicación específica de un dispositivo que admite conexiones de intérpretes de comandos SSH o Docker.

`ReadFromDevice`

Permite que los conjuntos de pruebas lean desde el puerto de serie de los dispositivos que admiten conexiones UART.

> **Note**
>
> Dado que IDT no gestiona las conexiones directas a los dispositivos que se realizan con información de acceso a los dispositivos procedente del contexto, recomendamos utilizar estos comandos de la API de interacción del dispositivo en los ejecutables de casos de prueba. Sin embargo, si estos comandos no cumplen los requisitos del caso de prueba,

puede recuperar la información de acceso al dispositivo desde el contexto de IDT y utilizarla para establecer una conexión directa con el dispositivo desde el conjunto de pruebas. Para establecer una conexión directa, recupere la información de los campos `device.connectivity` y `resource.devices.connectivity` del dispositivo que se está probando y de los dispositivos de recursos, respectivamente. Para obtener más información sobre cómo usar el contexto de IDT, consulte Utilice el contexto de IDT.

Interacción IDT

Los siguientes comandos permiten que sus conjuntos de pruebas se comuniquen con IDT.

`PollForNotifications`

Permite que los conjuntos de pruebas comprueben las notificaciones de IDT.

`GetContextValue` y `GetContextString`

Permite que los conjuntos de pruebas recuperen valores del contexto de IDT. Para obtener más información, consulte Utilice el contexto de IDT.

`SendResult`

Permite que los conjuntos de pruebas notifiquen los resultados de los casos de prueba a IDT. Este comando debe ejecutarse al final de cada caso de prueba en un conjunto de pruebas.

Interacción con el host

El siguiente comando permite que sus conjuntos de pruebas se comuniquen con el equipo principal.

`PollForNotifications`

Permite que los conjuntos de pruebas comprueben las notificaciones de IDT.

`GetContextValue` y `GetContextString`

Permite que los conjuntos de pruebas recuperen valores del contexto de IDT. Para obtener más información, consulte Utilice el contexto de IDT.

`ExecuteOnHost`

Permite que los conjuntos de pruebas ejecuten comandos en la máquina local y permite a IDT gestionar el ciclo de vida del ejecutable de caso de prueba.

Habilitar comandos CLI de IDT

El comando CLI de IDT `run-suite` proporciona varias opciones que permiten al ejecutor de pruebas personalizar la ejecución de las pruebas. Para permitir que los ejecutores de pruebas utilicen estas opciones para ejecutar su conjunto de pruebas personalizado, debe implementar la compatibilidad con la CLI de IDT. Si no implementa la compatibilidad, los ejecutores de pruebas podrán seguir ejecutándolas, pero algunas opciones de CLI no funcionarán correctamente. Para ofrecer una experiencia de cliente ideal, le recomendamos que implemente la compatibilidad con los siguientes argumentos para el comando `run-suite` en la CLI de IDT:

`timeout-multiplier`

Especifica un valor superior a 1,0 que se aplicará a todos los tiempos de espera durante la ejecución de las pruebas.

Los ejecutores de pruebas pueden usar este argumento para aumentar el tiempo de espera de los casos de prueba que desean ejecutar. Cuando un ejecutor de pruebas especifica este argumento en su comando `run-suite`, IDT lo usa para calcular el valor de la variable de entorno IDT_TEST_TIMEOUT y establece el campo `config.timeoutMultiplier` en el contexto de IDT. Para respaldar este argumento, debe hacer lo siguiente:

- En lugar de utilizar directamente el valor de tiempo de espera del archivo `test.json`, lea la variable de entorno IDT_TEST_TIMEOUT para obtener el valor de tiempo de espera calculado correctamente.
- Recupere el valor `config.timeoutMultiplier` del contexto de IDT y aplíquelo a los tiempos de espera prolongados.

Para obtener más información sobre cómo salir anticipadamente debido a eventos de tiempo de espera, consulte Especifique el comportamiento de salida.

`stop-on-first-failure`

Especifica que IDT debe dejar de ejecutar todas las pruebas si detecta un error.

Cuando un ejecutor de pruebas especifica este argumento en su comando `run-suite`, IDT dejará de ejecutar las pruebas en cuanto detecte un error. Sin embargo, si los casos de prueba se ejecutan en paralelo, esto puede generar resultados inesperados. Para implementar la compatibilidad, asegúrese de que si IDT detecta este evento, su lógica de pruebas indique a todos los casos de prueba en ejecución que se detengan, se limpien los recursos temporales y se notifique el resultado de la prueba a IDT. Para obtener más información sobre cómo salir a tiempo en caso de fracaso, consulte esta guía Especifique el comportamiento de salida.

`group-id` y `test-id`

Especifica que IDT debe ejecutar solo los grupos de pruebas o los casos de prueba seleccionados.

Los ejecutores de pruebas pueden usar estos argumentos con su comando `run-suite` para especificar el siguiente comportamiento de ejecución de pruebas:

- Ejecute todas las pruebas dentro de los grupos de pruebas especificados.
- Ejecute una selección de pruebas desde un grupo de pruebas específico.

Para respaldar estos argumentos, la máquina de estado de su conjunto de pruebas debe incluir un conjunto específico de estados `RunTask` y `Choice` en su máquina de estado. Si no utiliza una máquina de estado personalizada, la máquina de estado IDT predeterminada incluye los estados necesarios y no es necesario que realice ninguna acción adicional. Sin embargo, si utiliza una máquina de estado personalizada, use Ejemplo de máquina de estados: ejecute grupos de prueba seleccionados por el usuario como ejemplo para añadir los estados necesarios a su máquina de estado.

Para obtener más información sobre los comandos de la CLI de IDT, consulte Depurar y ejecutar conjuntos de pruebas personalizadas.

Escriba registros de eventos

Mientras se ejecuta la prueba, se envían datos a la consola `stdout` y `stderr` para escribir registros de eventos y mensajes de error en la consola. Para obtener más información sobre el formato de los mensajes de la consola, consulte Formato de mensajes de consola.

Cuando IDT termina de ejecutar el conjunto de pruebas, esta información también está disponible en el archivo `test_manager.log` ubicado en la carpeta *`<devicetester-extract-location>`*`/results/`*`<execution-id>`*`/logs`.

Puede configurar cada caso de prueba para que escriba los registros de la ejecución de la prueba, incluidos los registros del dispositivo que se está probando, en el archivo *`<group-id>_<test-id>`* ubicado en la carpeta *`<device-tester-extract-location>`*`/results/`*`execution-id`*`/logs`. Para ello, recupere la ruta del archivo de registro del contexto IDT con la consulta `testData.logFilePath`, cree un archivo en esa ruta y escriba el contenido que desee. IDT actualiza automáticamente la ruta en función del caso de prueba que se esté ejecutando. Si decide no crear el archivo de registro para un caso de prueba, no se generará ningún archivo para ese caso de prueba.

También puede configurar el ejecutable de texto para crear archivos de registro adicionales en la carpeta *<device-tester-extract-location>*/logs según sea necesario. Le recomendamos que especifique prefijos únicos para los nombres de los archivos de registro para que sus archivos no se sobrescriban.

Informe los resultados a IDT

IDT escribe los resultados de las pruebas en los archivos awsiotdevicetester_report.xml y *suite-name*_report.xml. Estos archivos de informes están ubicados en *<device-tester-extract-location>*/results/*<execution-id>*/. Ambos informes capturan los resultados de la ejecución del conjunto de pruebas. Para obtener más información sobre los esquemas que IDT utiliza para estos informes, consulte Revise los resultados y registros de las pruebas de IDT

Para rellenar el contenido del archivo *suite-name*_report.xml, debe utilizar el comando SendResult para informar de los resultados de las pruebas a IDT antes de que finalice la ejecución de la prueba. Si IDT no puede localizar los resultados de una prueba, emite un error para el caso de prueba. El siguiente extracto de Python muestra los comandos para enviar el resultado de una prueba a IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Si no informa los resultados a través de la API, IDT busca los resultados de las pruebas en la carpeta de artefactos de prueba. La ruta a esta carpeta se almacena en testData.testArtifactsPath archivado en el contexto de IDT. En esta carpeta, IDT utiliza el primer archivo XML ordenado alfabéticamente que localiza como resultado de la prueba.

Si la lógica de las pruebas produce resultados XML JUnit, puede escribir los resultados de la prueba en un archivo XML en la carpeta de artefactos para proporcionarlos directamente a IDT, en lugar de analizarlos y, a continuación, utilizar la API para enviarlos a IDT.

Si utiliza este método, asegúrese de que la lógica de las pruebas resuma con precisión los resultados de las pruebas y formatee el archivo de resultados con el mismo formato que el archivo *suite-name*_report.xml. IDT no realiza ninguna validación de los datos que usted proporciona, con las siguientes excepciones:

- IDT ignora todas las propiedades de la etiqueta testsuites. En su lugar, calcula las propiedades de la etiqueta a partir de los resultados de otros grupos de pruebas notificados.
- Debe haber al menos una etiqueta testsuite en testsuites.

Dado que IDT utiliza la misma carpeta de artefactos para todos los casos de prueba y no elimina los archivos de resultados entre las ejecuciones de las pruebas, este método también puede provocar informes erróneos si IDT lee el archivo incorrecto. Le recomendamos que utilice el mismo nombre para el archivo de resultados XML generado en todos los casos de prueba para sobrescribir los resultados de cada caso de prueba y asegurarse de que IDT pueda utilizar los resultados correctos. Si bien puede utilizar un enfoque mixto para la elaboración de informes en su conjunto de pruebas, es decir, utilizar un archivo de resultados XML para algunos casos de prueba y enviar los resultados a través de la API para otros, no recomendamos este enfoque.

Especifique el comportamiento de salida

Configure el ejecutable de texto para que siempre salga con un código de salida de 0, incluso si un caso de prueba informa de un fallo o un resultado de error. Utilice códigos de salida distintos de cero únicamente para indicar que un caso de prueba no se ha ejecutado o si el ejecutable del caso de prueba no ha podido comunicar ningún resultado a IDT. Cuando IDT recibe un código de salida distinto de cero, indica que el caso de prueba ha detectado un error que ha impedido su ejecución.

IDT podría solicitar o esperar que un caso de prueba deje de ejecutarse antes de que finalice en los siguientes eventos. Utilice esta información para configurar el ejecutable del caso de prueba para que detecte cada uno de estos eventos del caso de prueba:

Timeout (Tiempo de espera)

Se produce cuando un caso de prueba se ejecuta durante más tiempo que el valor de tiempo de espera especificado en el archivo `test.json`. Si el ejecutor de la prueba utilizó el argumento `timeout-multiplier` para especificar un multiplicador de tiempo de espera, IDT calcula el valor de tiempo de espera con el multiplicador.

Para detectar este evento, utilice la variable de entorno IDT_TEST_TIMEOUT. Cuando un ejecutor de pruebas lanza una prueba, IDT establece el valor de la variable de entorno IDT_TEST_TIMEOUT en el valor de tiempo de espera calculado (en segundos) y pasa la variable al ejecutable del caso de prueba. Puede leer el valor de la variable para configurar un temporizador adecuado.

Interrumpir

Se produce cuando el ejecutor de pruebas interrumpe IDT. Por ejemplo, pulsando Ctrl+C.

Como los terminales propagan las señales a todos los procesos secundarios, solo tiene que configurar un controlador de señales en sus casos de prueba para detectar las señales de interrupción.

Como alternativa, puede sondear periódicamente la API para comprobar el valor del booleano `CancellationRequested` en la respuesta de la API `PollForNotifications`. Cuando IDT recibe una señal de interrupción, establece el valor del booleano `CancellationRequested` en `true`.

Detención en el primer fallo

Se produce cuando un caso de prueba que se está ejecutando en paralelo con el caso de prueba actual falla y el ejecutor de la prueba utiliza el argumento `stop-on-first-failure` para especificar que IDT debe detenerse cuando encuentra algún fallo.

Para detectar este evento, puede sondear periódicamente la API para comprobar el valor del booleano `CancellationRequested` en la respuesta de la API al `PollForNotifications`.. Cuando IDT detecta un fallo y se configura para detenerse en el primer fallo, establece el valor del booleano `CancellationRequested` en `true`.

Cuando se produce alguno de estos eventos, IDT espera 5 minutos a que los casos de prueba que se estén ejecutando actualmente terminen de ejecutarse. Si todos los casos de prueba en ejecución no se cierran en 5 minutos, IDT obliga a detener cada uno de sus procesos. Si IDT no ha recibido los resultados de las pruebas antes de que finalicen los procesos, marcará los casos de prueba como agotados. Como práctica recomendada, debe asegurarse de que los casos de prueba realicen las siguientes acciones cuando detecten alguno de estos eventos:

1. Dejen de ejecutar la lógica de prueba normal.
2. Limpien todos los recursos temporales, como los artefactos de prueba del dispositivo que se está probando.
3. Informen el resultado de una prueba a IDT, como un fallo o un error en la prueba.
4. Salir

Utilice el contexto de IDT

Cuando IDT ejecuta un conjunto de pruebas, el conjunto de pruebas puede acceder a un conjunto de datos que se pueden utilizar para determinar cómo se ejecuta cada prueba. Estos datos se denominan contexto IDT. Por ejemplo, la configuración de los datos de usuario proporcionada por los ejecutores de pruebas en un archivo `userdata.json` se pone a disposición de los conjuntos de pruebas en el contexto de IDT.

El contexto IDT puede considerarse un documento JSON de solo lectura. Los conjuntos de pruebas pueden recuperar datos del contexto y escribirlos en él mediante tipos de datos JSON estándar, como objetos, matrices, números, etc.

Esquema de contexto

El contexto de IDT utiliza el siguiente formato:

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
    "testData": {
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

`config`

Información del archivo `config.json`. El campo `config` también contiene el siguiente campo adicional:

`config.timeoutMultiplier`

El multiplicador para cualquier valor de tiempo de espera utilizado por el conjunto de pruebas. El ejecutor de pruebas especifica este valor desde la CLI de IDT. El valor predeterminado es 1.

`device`

Información sobre el dispositivo seleccionado para la prueba. Esta información equivale al elemento de matriz `devices` del archivo `device.json` del dispositivo seleccionado.

`devicePool`

Información sobre el grupo de dispositivos seleccionado para la ejecución de la prueba. Esta información equivale al elemento de matriz del dispositivo en el nivel superior seleccionado en el archivo `device.json` para la matriz de dispositivos seleccionados.

`resource`

Información sobre los dispositivos de recursos del archivo `resource.json`.

`resource.devices`

Esta información equivale a la matriz `devices` definida en el archivo `resource.json`. Cada elemento `devices` incluye el siguiente campo adicional:

`resource.device.name`

El nombre del dispositivo de recursos. Este valor se establece en el valor `requiredResource.name` del archivo `test.json`.

`testData.awsCredentials`

Las credenciales AWS utilizadas por la prueba para conectarse a la nube AWS. Esta información se obtiene del archivo `config.json`.

`testData.logFilePath`

La ruta al archivo de registro en el que el caso de prueba escribe los mensajes de registro. El conjunto de pruebas crea este archivo si no existe.

`userData`

Información proporcionada por el ejecutor de la prueba en el archivo `userdata.json`.

Acceda a los datos en el contexto

Puede consultar el contexto mediante la notación JSONPath de sus archivos JSON y de su ejecutable de texto con las API `GetContextValue` y`GetContextString`. La sintaxis de las cadenas JSONPath para acceder al contexto IDT varía de la siguiente manera:

- En `suite.json` y `test.json`, se usa `{{`*`query`*`}}` Es decir, no utilice el elemento raíz `$.` para iniciar la expresión.
- En `statemachine.json`, usted usa `{{$.`*`query`*`}}`.
- En los comandos de la API, se utiliza *`query`* o `{{$.`*`query`*`}}`, según el comando. Para obtener más información, consulte la documentación en línea en los SDK.

En la siguiente tabla se describen los operadores de una expresión JSONPath típica:

Operator	Description
`$`	The root element. Because the top-level context value for IDT is an object, you will typically use `$.` to start your queries.
`.childName`	Accesses the child element with name `childName` from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the `awsRegion` value in the `config` object is `$.config.awsRegion` .
`[start:end]`	Filters elements from an array, retrieving items beginning from the `iniciar` index and going up to the `END` index, both inclusive.

Operator	Description
`[index1, index2, ... , indexN]`	Filters elements from an array, retrieving items from only the specified indices.
`[?(expr)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

Para crear expresiones de filtro, utilice la siguiente sintaxis:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

En esta sintaxis:

- `jsonpath` es un JSONPath que utiliza la sintaxis JSON estándar.
- `value` es cualquier valor personalizado que utilice la sintaxis JSON estándar.
- `operator` es uno de los siguientes operadores:
  - < (Menor que)
  - <= (Menor o igual que)
  - == (Igual que)

    Si el JSONPath o el valor de la expresión es un valor matricial, booleano o de objeto, este es el único operador binario compatible que puede utilizar.
  - >= (Mayor o igual que)
  - > (Mayor que)
  - =~ (La expresión regular coincide). Para usar este operador en una expresión de filtro, el JSONPath o el valor del lado izquierdo de la expresión debe dar como resultado una cadena y el lado derecho debe ser un valor de patrón que siga la sintaxis RE2.

Puede utilizar consultas JSONPath de la forma {{*query*}} como cadenas de marcador de posición dentro de los campos `args` y `environmentVariables` en los archivos `test.json` y dentro de los campos `environmentVariables` en los archivos `suite.json`. IDT realiza una búsqueda contextual y rellena los campos con el valor evaluado de la consulta. Por ejemplo, en el archivo `suite.json`, puede utilizar cadenas de marcadores de posición para especificar los valores de

las variables de entorno que cambian con cada caso de prueba, e IDT rellenará las variables de entorno con el valor correcto para cada caso de prueba. Sin embargo, cuando se utilizan cadenas de marcadores de posición en `test.json` y en los archivos `suite.json` y, las consultas tienen en cuenta las siguientes consideraciones:

- Debe escribir en minúsculas cada vez que aparezca la clave `devicePool` en la consulta. En su lugar, utilice las API `devicepool`.
- Para las matrices, solo puede usar matrices de cadenas. Además, las matrices utilizan un formato `item1, item2,...,itemN` no estándar. Si la matriz contiene solo un elemento, se serializa como `item`, lo que la hace indistinguible de un campo de cadena.
- No puede utilizar marcadores de posición para recuperar objetos del contexto.

Debido a estas consideraciones, le recomendamos que, siempre que sea posible, utilice la API para acceder al contexto en su lógica de prueba en lugar de cadenas de marcadores de posición en `test.json` y en los archivos `suite.json`. Sin embargo, en algunos casos puede ser más conveniente utilizar marcadores de posición de JSONPath para recuperar cadenas individuales y configurarlas como variables de entorno.

Configure los ajustes para los ejecutores de pruebas

Para ejecutar conjuntos de pruebas personalizados, los ejecutores de pruebas deben configurar sus ajustes en función del conjunto de pruebas que desean ejecutar. Los ajustes se especifican en función de las plantillas del archivo de configuración JSON que se encuentran en la carpeta *`<device-tester-extract-location>`*`/configs/`. Si es necesario, los ejecutores de las pruebas también deben configurar las credenciales de AWS que IDT utilizará para conectarse a la nube AWS.

Como redactor de pruebas, necesitará configurar estos archivos para depurar su conjunto de pruebas. Debe proporcionar instrucciones a los ejecutores de pruebas para que puedan configurar los siguientes ajustes según sea necesario para ejecutar sus conjuntos de pruebas.

Configurar device.json

El archivo `device.json` contiene información sobre los dispositivos en los que se ejecutan las pruebas (por ejemplo, dirección IP, información de acceso, sistema operativo y arquitectura de la CPU).

Los ejecutores de pruebas pueden proporcionar esta información mediante el siguiente archivo `device.json` de plantilla que se encuentra en la carpeta *<device-tester-extract-location>*`/configs/`.

```
[
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",
```

```
                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID alfanumérico definido por el usuario que identifica de forma única una colección de dispositivos que se conoce como grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben tener idéntico hardware. Al ejecutar un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo. Se utilizan varios dispositivos para ejecutar diferentes pruebas.

`sku`

Un valor alfanumérico que identifica de forma única el dispositivo a prueba. El SKU se utiliza para realizar un seguimiento de los dispositivos cualificados.

> **Note**
>
> Si desea enumerar la placa en el catálogo de dispositivos de AWS Partner, el SKU que especifique aquí debe coincidir con el SKU que utilice en el proceso de publicación.

`features`

Opcional. Una matriz que contenga las características compatibles del dispositivo. Las características del dispositivo son valores definidos por el usuario que se configuran en el conjunto de pruebas. Debe proporcionar a los participantes en las pruebas información sobre los nombres y valores de las características que desee incluir en el archivo `device.json`. Por ejemplo, si quiere probar un dispositivo que funciona como servidor MQTT para otros dispositivos, puede configurar la lógica de prueba para validar los niveles admitidos específicos para una característica denominada `MQTT_QOS`. Los ejecutores de pruebas proporcionan el nombre de esta característica y establecen el valor de la función en los niveles de QOS compatibles con su dispositivo. Puede recuperar la información proporcionada desde el contexto IDT con la

consulta `devicePool.features`, o desde el contexto máquina de estado con la consulta `pool.features`.

`features.name`

El nombre de la característica.

`features.value`

Los valores de las características compatibles.

`features.configs`

Los ajustes de configuración de la característica, si son necesarios.

`features.config.name`

El nombre del ajuste de configuración.

`features.config.value`

Los valores de configuración admitidos.

`devices`

Un conjunto de dispositivos en el grupo a probar. Se requiere al menos un dispositivo.

`devices.id`

Un identificador único y definido por el usuario para el dispositivo que se está probando.

`connectivity.protocol`

El protocolo de comunicación que se usará para la comunicación con este dispositivo. Cada dispositivo de un grupo debe usar el mismo protocolo.

Actualmente, los únicos valores que se admiten son `ssh` y `uart` para dispositivos físicos, y `docker` para contenedores de Docker.

`connectivity.ip`

La dirección IP del dispositivo que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.port`

Opcional. El número de puerto a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.password`

La contraseña se utiliza para iniciar sesión en el dispositivo que se va a probar.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo que se está probando.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`connectivity.auth.credentials.user`

El nombre de usuario para iniciar sesión en el dispositivo que se está probando.

`connectivity.serialPort`

Opcional. El puerto serie al que está conectado el dispositivo.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `uart`.

`connectivity.containerId`

El ID de contenedor o el nombre del contenedor de Docker que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `docker`.

`connectivity.containerUser`

Opcional. El nombre del usuario a utilizar dentro del contenedor. El valor predeterminado es el usuario proporcionado en el Dockerfile.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `docker`.

> **Note**
>
> Para comprobar si los ejecutores de las pruebas configuran una conexión de dispositivo incorrecta para una prueba, puede recupera `pool.Devices[0].Connectivity.Protocol` del contexto de la máquina de estados y realizar una comparación con el valor esperado en un estado `Choice`. Si se utiliza un protocolo incorrecto, imprima un mensaje con el estado `LogMessage` y haga la transición al estado `Fail`.
> Como alternativa, puede utilizar un código de gestión de errores para informar de un fallo en la prueba si el tipo de dispositivo es incorrecto.

(Opcional) Configuración de userdata.json

El archivo `userdata.json` contiene cualquier información adicional que requiera un conjunto de pruebas, pero que no esté especificada en el archivo `device.json`. El formato de este archivo depende del `userdata_scheme.json`archivo definido en el conjunto de pruebas. Si es un redactor de pruebas, asegúrese de proporcionar esta información a los usuarios que ejecutarán los conjuntos de pruebas que escriba.

(Opcional) Configuración de resource.json

El archivo `resource.json` contiene información sobre los dispositivos que se utilizarán como dispositivos de recursos. Los dispositivos de recursos son dispositivos que se requieren para probar ciertas capacidades de un dispositivo que se está probando. Por ejemplo, para probar la capacidad Bluetooth de un dispositivo, puedes usar un dispositivo de recursos para comprobar si el dispositivo se puede conectar correctamente a él. Los dispositivos de recursos son opcionales y puede necesitar tantos dispositivos de recursos como necesite. Como redactor de pruebas, utiliza el archivo test.json para definir las funciones del dispositivo de recursos que se requieren

para una prueba. A continuación, los ejecutores de pruebas utilizan el archivo `resource.json` para proporcionar un conjunto de dispositivos de recursos que tengan las funciones necesarias. Asegúrese de proporcionar esta información a los usuarios que vayan a ejecutar los conjuntos de pruebas que escriba.

Los ejecutores de pruebas pueden proporcionar esta información mediante el siguiente archivo `resource.json` de plantilla que se encuentra en la carpeta *<device-tester-extract-location>*`/configs/`.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-value>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
```

```
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID alfanumérico definido por el usuario que identifica de forma única una colección de dispositivos que se conoce como grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben tener idéntico hardware. Al ejecutar un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo. Se utilizan varios dispositivos para ejecutar diferentes pruebas.

`features`

Opcional. Una matriz que contenga las características compatibles del dispositivo. La información requerida en este campo se define en los archivos test.json del conjunto de pruebas y determina qué pruebas se van a ejecutar y cómo se van a ejecutar. Si el conjunto de pruebas no requiere ninguna característica, este campo no es obligatorio.

`features.name`

El nombre de la característica.

`features.version`

La versión de la característica.

`features.jobSlots`

Configuración para indicar cuántas pruebas pueden utilizar el dispositivo simultáneamente. El valor predeterminado es 1.

`devices`

Un conjunto de dispositivos en el grupo a probar. Se requiere al menos un dispositivo.

`devices.id`

Un identificador único y definido por el usuario para el dispositivo que se está probando.

`connectivity.protocol`

El protocolo de comunicación que se usará para la comunicación con este dispositivo. Cada dispositivo de un grupo debe usar el mismo protocolo.

Actualmente, los únicos valores que se admiten son `ssh` y `uart` para dispositivos físicos, y `docker` para contenedores de Docker.

`connectivity.ip`

La dirección IP del dispositivo que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.port`

Opcional. El número de puerto a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.password`

La contraseña se utiliza para iniciar sesión en el dispositivo que se va a probar.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo que se está probando.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`connectivity.auth.credentials.user`

El nombre de usuario para iniciar sesión en el dispositivo que se está probando.

`connectivity.serialPort`

Opcional. El puerto serie al que está conectado el dispositivo.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `uart`.

`connectivity.containerId`

El ID de contenedor o el nombre del contenedor de Docker que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `docker`.

`connectivity.containerUser`

Opcional. El nombre del usuario a utilizar dentro del contenedor. El valor predeterminado es el usuario proporcionado en el Dockerfile.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `docker`.

(Opcional) Configuración de config.json

El archivo `config.json` contiene información de configuración para IDT. Por lo general, los ejecutores de pruebas no necesitarán modificar este archivo excepto para proporcionar sus credenciales de usuario de AWS para IDT y, opcionalmente, una región AWS. Si se proporcionan las credenciales de AWS con los permisos necesarios, AWS IoT Device Tester recopila y envía las métricas de uso a AWS. Se trata de una característica opcional y se utiliza para mejorar la funcionalidad de IDT. Para obtener más información, consulte Métricas de uso de IDT.

Los ejecutores de pruebas pueden configurar sus credenciales de AWS de una de las siguientes maneras:

- Archivo de credenciales

  IDT utiliza el mismo archivo de credenciales que la AWS CLI. Para obtener más información, consulte Archivos de configuración y credenciales.

  La ubicación del archivo de credenciales varía en función del sistema operativo que utilice:

  - macOS, Linux: `~/.aws/credentials`
  - Windows: `C:\Users\UserName\.aws\credentials`
- Variables de entorno

  Las variables de entorno son las variables que mantiene el sistema operativo y utilizan los comandos del sistema. Las variables definidas durante una sesión de SSH no están disponibles una vez cerrada la sesión. IDT pude usar las variables de entorno `AWS_ACCESS_KEY_ID` y `AWS_SECRET_ACCESS_KEY` para almacenar sus credenciales de AWS.

  Para establecer estas variables en Linux, MacOS, o Unix, utilice export:

  ```
  export AWS_ACCESS_KEY_ID=<your_access_key_id>
  export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
  ```

  Para establecer estas variables en Windows, utilice set:

  ```
  set AWS_ACCESS_KEY_ID=<your_access_key_id>
  set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
  ```

Para configurar las credenciales de AWS para IDT, los ejecutores de pruebas editan la sección `auth` del archivo `config.json` ubicado en la carpeta *<device-tester-extract-location>*`/configs/`.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
```

```
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

> **Note**
>
> Todas las rutas de este archivo se definen en relación con *<device-tester-extract-location>*.

`log.location`

La ruta a la carpeta de registros en *<device-tester-extract-location>*.

`configFiles.root`

La ruta a la carpeta que contiene los archivos de configuración.

`configFiles.device`

La ruta del archivo `device.json`.

`testPath`

La ruta a la carpeta que contiene los conjuntos de pruebas.

`reportPath`

La ruta a la carpeta que contendrá los resultados de las pruebas después de que IDT ejecute un conjunto de pruebas.

`awsRegion`

Opcional. La región AWS que utilizarán los conjuntos de pruebas. Si no se establece, los conjuntos de pruebas utilizarán la región predeterminada especificada en cada conjunto de pruebas.

`auth.method`

El método que utiliza IDT para recuperar las credenciales de AWS. Los valores admitidos son `file` para recuperar las credenciales de un archivo de credenciales y `environment` para recuperar las credenciales mediante variables de entorno.

`auth.credentials.profile`

El perfil de credenciales que se va a utilizar del archivo de credenciales. Esta propiedad solo se aplica si `auth.method` está establecido en `file`.

Depurar y ejecutar conjuntos de pruebas personalizadas

Una vez establecida la configuración requerida, IDT puede ejecutar su conjunto de pruebas. El tiempo de ejecución del conjunto de pruebas completa depende del hardware y de la composición del conjunto de pruebas. Como referencia, se tarda aproximadamente 30 minutos en completar el conjunto AWS IoT Greengrass de pruebas completo en una unidad Raspberry Pi 3B.

Mientras escribe su conjunto de pruebas, puede usar IDT para ejecutarla en modo de depuración, comprobar el código antes de ejecutarla o proporcionárselo a los ejecutores de pruebas.

Ejecuta IDT en modo de depuración

Como los conjuntos de pruebas dependen de IDT para interactuar con los dispositivos, proporcionar el contexto y recibir los resultados, no puede simplemente depurar sus suites de prueba en un IDE sin ninguna interacción con IDT. Para ello, la CLI de IDT proporciona el comando `debug-test-suite` que permite ejecutar IDT en modo de depuración. Ejecute el siguiente comando para ver las opciones disponibles para `debug-test-suite`:

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

Cuando se ejecuta IDT en modo de depuración, IDT no inicia realmente el conjunto de pruebas ni ejecuta la máquina de estados, sino que interactúa con el IDE para responder a las solicitudes realizadas desde el conjunto de pruebas que se ejecuta en el IDE e imprime los registros en la consola. IDT no agota el tiempo de espera y espera a salir hasta que se interrumpa manualmente. En el modo de depuración, IDT tampoco ejecuta la máquina de estados y no generará ningún archivo de informe. Para depurar su conjunto de pruebas, debe usar su IDE para proporcionar cierta información que IDT suele obtener de los archivos JSON de configuración. Asegúrese de que dispone de la siguiente información:

- Variables de entorno y argumentos para cada prueba. IDT no leerá esta información de `test.json` ni `suite.json`.
- Argumentos para seleccionar los dispositivos de recursos. IDT no leerá esta información de `test.json`.

Para depurar los conjuntos de pruebas, complete los pasos siguientes:

1. Cree los archivos de configuración necesarios para ejecutar el conjunto de pruebas. Por ejemplo, si su conjunto de pruebas requiere `device.json`, `resource.json`, y `user data.json`, asegúrese de configurarlos todos según sea necesario.
2. Ejecute el siguiente comando para poner IDT en modo de depuración y seleccione los dispositivos necesarios para ejecutar la prueba.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   Tras ejecutar este comando, IDT espera las solicitudes de el conjunto de pruebas y, a continuación, responde a ellas. IDT también genera las variables de entorno que se requieran para el proceso de casos del IDT Client SDK.
3. En su IDE, utilice la configuración `run` o `debug` para hacer lo siguiente:

   a. Establezca los valores de las variables de entorno generadas por IDT.

   b. Establezca el valor de cualquier variable o argumento de entorno que haya especificado en el archivo `test.json` y `suite.json`.

   c. Establezca los puntos de interrupción según sea necesario.
4. Ejecute el conjunto de pruebas en su IDE.

   Puede depurar y volver a ejecutar el conjunto de pruebas tantas veces como sea necesario. El tiempo de espera de IDT no se agota en el modo de depuración.
5. Una vez completada la depuración, interrumpa IDT para salir del modo de depuración.

Comandos CLI de IDT para ejecutar pruebas

En las secciones siguientes se describen los comandos de la herramienta IDT CLI.

IDT v4.0.0

`help`

Enumera información acerca del comando especificado.

`list-groups`

Muestra los grupos de un conjunto de prueba determinado.

`list-suites`

Muestra los conjuntos de prueba disponibles.

`list-supported-products`

Enumera los productos compatibles con su versión de IDT, en este caso las versionesAWS IoT Greengrass de cualificaciones AWS IoT Greengrass y las versiones del conjunto de pruebas para la versión actual de IDT.

`list-test-cases`

Enumera los casos de prueba en un grupo de prueba determinado. Se admite la siguiente opción:

- `group-id`. El grupo de prueba que se va a buscar. Esta opción es necesaria y debe especificar un solo grupo.

`run-suite`

Ejecuta un conjunto de pruebas en un grupo de dispositivos. A continuación se muestran algunas opciones que suelen utilizarse:

- `suite-id`. La versión del conjunto de pruebas que se va a ejecutar. Si no se especifica, IDT utiliza la versión más reciente de la carpeta `tests`.
- `group-id`. Los grupos de prueba que se van a ejecutar, como una lista separada por comas. Si no se especifica, IDT ejecuta todos los grupos de prueba del conjunto de pruebas.
- `test-id`. Los casos de prueba a ejecutar, como lista separada por comas. Cuando se especifique, `group-id` debe especificar un solo grupo.
- `pool-id`. El grupo de dispositivos que se va a probar. Las ejecuciones deben especificar un grupo si tiene varios grupos de dispositivos definidos en el archivo `device.json`.

- `timeout-multiplier`. Configura IDT para modificar el tiempo de espera de ejecución de la prueba especificado en el archivo `test.json` para una prueba con un multiplicador definido por el usuario.
- `stop-on-first-failure`. Configura IDT para detener la ejecución en el primer error. Esta opción debe utilizarse con `group-id` para depurar los grupos de prueba especificados.
- `userdata`. Establece el archivo que contiene la información sobre los datos del usuario necesarios para el conjunto de pruebas. Esto solo es necesario si `userdataRequired` se establece en verdadero en el archivo `suite.json` del conjunto de pruebas.

Para obtener más información acerca de `run-suite` las opciones, utilice la opción `help`:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

`debug-test-suite`

Ejecute el conjunto de pruebasen modo de depuración. Para obtener más información, consulte Ejecuta IDT en modo de depuración.

Revise los resultados y registros de las pruebas de IDT

En esta sección, se describe el formato en el que IDT genera los registros de la consola y los informes de las pruebas.

Formato de mensajes de consola

AWS IoT Device Tester utiliza un formato estándar para imprimir mensajes en la consola cuando inicia un conjunto de pruebas. En el fragmento siguiente, se muestra un ejemplo de mensajes de consola generados por IDT.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La mayoría de los mensajes de consola constan de los siguientes campos:

`time`

Una marca de tiempo completa conforme a la norma ISO 8601 para el evento registrado.

`level`

El nivel de mensaje del evento registrado. Normalmente, el nivel del mensaje registrado es uno de los siguientes:`info`, `warn` o `error`. IDT emite un mensaje `panic` o `fatal` si detecta un evento esperado que provoca su cierre anticipado.

`msg`

El mensaje registrado.

`executionId`

Una cadena de identificación única para el proceso de IDT actual. Este identificador se utiliza para diferenciar entre ejecuciones de IDT individuales.

Los mensajes de consola generados a partir de un conjunto de pruebas proporcionan información adicional sobre el dispositivo que se está probando y el conjunto de pruebas, el grupo de pruebas y los casos de prueba que ejecuta IDT. En el fragmento siguiente, se muestra un ejemplo de mensajes de consola generados por un conjunto de pruebas.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La parte específica del mensaje de la consola para el conjunto de pruebas contiene los siguientes campos:

`suiteId`

El nombre del conjunto de pruebas que se está ejecutando actualmente.

`groupId`

La identificación del grupo de pruebas que se está ejecutando actualmente.

`testCaseId`

La identificación del caso de prueba que se está ejecutando actualmente.

`deviceId`

Un identificador del dispositivo que se está probando y que el caso de prueba está utilizando.

Para imprimir un resumen de la prueba en la consola cuando un IDT termine de ejecutar una prueba, debe incluir un estado de `Report` en la máquina de estado. El resumen de la prueba contiene información sobre el conjunto de pruebas, los resultados de las pruebas de cada grupo que se ejecutó y las ubicaciones de los registros y archivos de informes generados. El siguiente ejemplo muestra un mensaje resumen de prueba.

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
---------------------------------
Test Groups:
    GroupA:         PASSED
    GroupB:         FAILED
---------------------------------
Failed Tests:
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
---------------------------------
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

Esquema de informes de AWS IoT Device Tester

`awsiotdevicetester_report.xml` es un informe firmado que contiene la siguiente información:

- La versión de IDT.
- La versión del conjunto de pruebas.
- La firma y la clave del informe utilizadas para firmarlo.
- El SKU de dispositivos y el grupo de dispositivos especificado en el archivo `device.json`.
- La versión del producto y las características del dispositivo que se probaron.
- El resumen de agregación de los resultados de las pruebas. Esta información es la misma que la que figura en el archivo *`suite-name`*`_report.xml`.

```
<apnreport>
```

```
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

El archivo `awsiotdevicetester_report.xml` contiene una etiqueta `<awsproduct>` que tiene información sobre el producto que se está probando y las características del producto que se han validado después de ejecutar un conjunto de pruebas.

Atributos que se utilizan en la etiqueta **`<awsproduct>`**

`name`

El nombre del producto que se está probando.

`version`

La versión del producto que se está probando.

`features`

Las características validadas. Las funciones marcadas como `required` son necesarias para que el conjunto de pruebas valide el dispositivo. En el siguiente fragmento se muestra cómo aparece esta información en el archivo `awsiotdevicetester_report.xml`:

```
<feature name="ssh" value="supported" type="required"></feature>
```

Las funciones marcadas como `optional` no son obligatorias para la validación. Los siguientes fragmentos muestran características opcionales:

```
<feature name="hsi" value="supported" type="optional"></feature>

<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Esquema del informe del conjunto de pruebas

El informe *`suite-name`*`_Result.xml` está en formato XML JUnit. Puede integrarlo en plataformas de integración/implementación continua como Jenkins, Bamboo, etc. El informe contiene un resumen global de los resultados de las pruebas.

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
```

```
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
        </testcase>
        <!--error-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

La sección de informe tanto en `awsiotdevicetester_report.xml` como en *`suite-name`*`_report.xml` enumera las pruebas que se ejecutaron y los resultados.

La primera etiqueta XML `<testsuites>` contiene el resumen de la ejecución de las pruebas. Por ejemplo:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Atributos que se utilizan en la etiqueta **`<testsuites>`**

`name`

El nombre del grupo de prueba.

`time`

El tiempo, en segundos, que se ha tardado en ejecutar el conjunto de pruebas.

`tests`

El número de pruebas ejecutadas.

`failures`

El número de pruebas que se ejecutaron, pero que no se superaron.

`errors`

El número de pruebas que IDT no ha podido ejecutar.

`disabled`

Este atributo no se utiliza y se puede omitir.

Si se producen errores en pruebas, puede identificar la prueba fallido revisando las etiquetas XML `<testsuites>`. Las etiquetas XML `<testsuite>` dentro de la etiqueta `<testsuites>` muestran el resumen del resultado de la prueba de un grupo de prueba. Por ejemplo:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

El formato es similar a la etiqueta `<testsuites>`, pero con un atributo `skipped` que no se utiliza y que se puede pasar por alto. Dentro de cada etiqueta XML `<testsuite>`, hay etiquetas `<testcase>` para cada prueba ejecutada para un grupo de prueba. Por ejemplo:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

Atributos que se utilizan en la etiqueta **`<testcase>`**

`name`

El nombre de la prueba.

`attempts`

El número de veces que IDT ha ejecutado el caso de prueba.

Cuando una prueba genera un error o si se produce un error, las etiquetas `<failure>` o `<error>` se añaden a la etiqueta `<testcase>` con información para la resolución de problemas. Por ejemplo:

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

Métricas de uso de IDT

Si proporciona AWS credenciales con los permisos necesarios, AWS IoT Device Tester recopila y envía las métricas de uso a. AWS Se trata de una característica opcional que se utiliza para mejorar la funcionalidad de IDT. IDT recopila información como la siguiente:

- El Cuenta de AWS ID utilizado para ejecutar IDT
- Los comandos de la CLI de IDT para ejecutar pruebas
- El conjunto de pruebas que se ejecutan
- Los conjuntos de pruebas de la carpeta *< device-tester-extract-location >*
- La cantidad de dispositivos configurados en el grupo de dispositivos
- Los nombres de casos de prueba y los tiempos de ejecución
- La información sobre los resultados de las pruebas, por ejemplo, si se han superado, si han fallado, si se han encontrado errores o si se han omitido
- Las características del producto probadas
- El comportamiento de salida de IDT, como salidas inesperadas o anticipadas

Toda la información que IDT envía también se registra en un archivo `metrics.log` de la carpeta *<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Puede consultar el archivo de registro para ver la información recopilada durante la ejecución de una prueba. Este archivo se genera solo si elige recopilar métricas de uso.

Para deshabilitar la recopilación de métricas, no es necesario que realice ninguna acción adicional. Simplemente no almacene sus AWS credenciales y, si AWS las tiene almacenadas, no configure el archivo `config.jso` n para acceder a ellas.

Configure sus AWS credenciales

Si aún no tiene una Cuenta de AWS, debe crear una. Si ya tiene una Cuenta de AWS, solo tiene que configurar los permisos necesarios para su cuenta para que IDT pueda enviarle las métricas de uso AWS en su nombre.

Paso 1: Crea una Cuenta de AWS

En este paso, cree y configure una Cuenta de AWS. Si ya tiene uno Cuenta de AWS, vaya athe section called “Paso 2: Configurar los permisos de IDT”.

Inscríbase en una Cuenta de AWS

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

Para suscribirte a una Cuenta de AWS

1. Abra https://portal.aws.amazon.com/billing/signup.

2. Siga las instrucciones que se le indiquen.

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

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

AWS te envía un correo electrónico de confirmación una vez finalizado el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en https://aws.amazon.com/ y seleccionando Mi cuenta.

Creación de un usuario con acceso administrativo

Después de crear un usuario administrativo Cuenta de AWS, asegúrelo Usuario raíz de la cuenta de AWS AWS IAM Identity Center, habilite y cree un usuario administrativo para no usar el usuario root en las tareas diarias.

Proteja su Usuario raíz de la cuenta de AWS

1. Inicie sesión AWS Management Consolecomo propietario de la cuenta seleccionando el usuario root e introduciendo su dirección de Cuenta de AWS correo electrónico. En la siguiente página, escriba su contraseña.

   Para obtener ayuda para iniciar sesión con el usuario raíz, consulte Signing in as the root user en la Guía del usuario de AWS Sign-In .

2. Active la autenticación multifactor (MFA) para el usuario raíz.

   Para obtener instrucciones, consulte Habilitar un dispositivo MFA virtual para el usuario Cuenta de AWS raíz (consola) en la Guía del usuario de IAM.

Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

   Consulte las instrucciones en Activar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center .

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

   Para ver un tutorial sobre su uso Directorio de IAM Identity Center como fuente de identidad, consulte Configurar el acceso de los usuarios con la configuración predeterminada Directorio de IAM Identity Center en la Guía del AWS IAM Identity Center usuario.

Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

  Para obtener ayuda para iniciar sesión con un usuario del Centro de identidades de IAM, consulte Iniciar sesión en el portal de AWS acceso en la Guía del AWS Sign-In usuario.

Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

   Para conocer las instrucciones, consulte Create a permission set en la Guía del usuario de AWS IAM Identity Center .
2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

   Para conocer las instrucciones, consulte Add groups en la Guía del usuario de AWS IAM Identity Center .

Paso 2: Configurar los permisos de IDT

En este paso, configure los permisos que IDT utiliza para ejecutar las pruebas y recopilar datos de uso de IDT. Puede usar AWS Management Console o AWS Command Line Interface (AWS CLI) para crear una política de IAM y un usuario para IDT y, a continuación, adjuntar políticas al usuario.

- Configuración de permisos para IDT (consola)
- Configuración de permisos para IDT (AWS CLI)

Configuración de permisos de IDT (consola)

Siga estos pasos para usar la consola para configurar permisos para IDT para AWS IoT Greengrass.

1. Inicie sesión en la consola de IAM.
2. Crear una política administrada que conceda permisos para crear roles con permisos específicos.

   a. En el panel de navegación, seleccione Políticas y, a continuación, Crear política.

   b. En la pestaña JSON, reemplace el contenido del marcador de posición por la política siguiente.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

   c. Elija Siguiente: Etiquetas.

   d. Elija Siguiente: Revisar.

   e. En Nombre, escriba **`IDTUsageMetricsIAMPermissions`**. En Summary (Resumen), revise los permisos concedidos por la política.

   f. Elija Crear política.

3. Cree un usuario de IAM y adjunte los permisos al usuario.

   a. Cree un usuario de IAM. Siga los pasos del 1 al 5 en Creación de usuarios de IAM (consola) en la Guía del usuario de IAM. Si ya ha creado un usuario de IAM, pase directamente al siguiente paso.

   b. Adjunte los permisos a su usuario de IAM:

      i. En la página Establecer permisos, elija Adjuntar políticas existentes directamente.

ii. Busque la política UsageMetricsIAMPermissions de IDT que creó en el paso anterior. Seleccione la casilla de verificación.

c. Elija Siguiente: etiquetas.

d. Elija Next: Review (Siguiente: revisar) para ver un resumen de sus opciones.

e. Seleccione la opción Crear un usuario.

f. Para ver las claves de acceso del usuario (ID de clave de acceso y claves de acceso secretas), elija Show (Mostrar) junto a la contraseña y la clave de acceso. Para guardar las claves de acceso, elija Download.csv (Descargar archivo .csv) y, a continuación, guarde el archivo en un lugar seguro. Utilizará esta información más adelante para configurar el archivo de credenciales. AWS

Configuración de permisos de IDT (AWS CLI)

Siga estos pasos AWS CLI para configurar los permisos de IDT para AWS IoT Greengrass. Si ya ha configurado permisos en la consola, vaya a the section called “Configure su dispositivo para ejecutar pruebas de IDT” o the section called “Opcional: Configuración del contenedor Docker”.

1. En su ordenador, instale y configure el AWS CLI si aún no está instalado. Siga los pasos que se indican en Instalación de la AWS CLI en la Guía del usuario de la AWS Command Line Interface .

   > **Note**
   >
   > AWS CLI Se trata de una herramienta de código abierto que puede utilizar para interactuar con los AWS servicios desde el shell de la línea de comandos.

2. Cree la siguiente política gestionada por el cliente que conceda permisos para gestionar el IDT y las funciones. AWS IoT Greengrass

   Linux, macOS, or Unix

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document '{
       "Version": "2012-10-17",
       "Statement": [
           {
   ```

```
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document
                                          '{\"Version\": \"2012-10-17\",
 \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-
tester:SendMetrics\"], \"Resource\": \"*\"}]}'
```

> **Note**
>
> Este paso incluye un ejemplo de símbolo del sistema de Windows porque utiliza una sintaxis JSON diferente a la de los comandos de terminal Linux, macOS o Unix.

3. Cree un usuario de IAM y adjunte los permisos requeridos por IDT para AWS IoT Greengrass.

   a. Cree un usuario de IAM.

   ```
   aws iam create-user --user-name user-name
   ```

   b. Adjunte la política `IDTUsageMetricsIAMPermissions` que ha creado a su nuevo usuario de IAM. Sustituya *el nombre de usuario* por su nombre de usuario de IAM y *<account-id>* en el comando con la identificación de su Cuenta de AWS.

   ```
   aws iam attach-user-policy --user-name user-name --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

4. Cree una clave de acceso secreta para el usuario.

```
aws iam create-access-key --user-name user-name
```

Almacene la salida en una ubicación segura. Utilizará esta información más adelante para configurar el archivo de AWS credenciales.

Proporcione AWS las credenciales a IDT

Para permitir que IDT acceda a sus AWS credenciales y envíe las métricas a ellas AWS, haga lo siguiente:

1. Guarde las AWS credenciales de su usuario de IAM como variables de entorno o en un archivo de credenciales:

   a. Para usar variables de entorno, ejecute el siguiente comando:

   ```
   AWS_ACCESS_KEY_ID=access-key
   AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

   b. Para utilizar el archivo de credenciales, añada la siguiente información a `.aws/credentials file:`

   ```
   [profile-name]
   aws_access_key_id=access-key
   aws_secret_access_key=secret-access-key
   ```

2. Configure la sección `auth` del archivo `config.json`. Para obtener más información, consulte (Opcional) Configuración de config.json.

Solución de problemas de IDT para AWS IoT Greengrass

IDT para AWS IoT Greengrass escribe estos errores en varias ubicaciones en función del tipo de errores. Los errores se escriben en la consola, en los archivos de registro y en los informes de prueba.

Códigos de error

En la siguiente tabla se muestran los códigos de error generados por IDT para AWS IoT Greengrass.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
101	InternalError	Se ha producido un error interno.	Compruebe los registros en el directorio *<device-tester-extract-location>* `/results`. Si no puede solucionar el problema, póngase en contacto con AWSDeveloper Support.
102	TimeoutError	La prueba no se puede completar en un período de tiempo limitado. Esto puede ocurrir si: • La conexión de red entre la máquina que se prueba y el dispositivo es lenta (por ejemplo, si utiliza una red VPN). • Una red lenta retrasa la comunicación entre el dispositivo y la nube. •	• Compruebe la conexión de la red y la velocidad. • Asegúrese de que no ha modificado ningún archivo en el directorio `/test`. • Intente ejecutar el grupo de pruebas que ha generado el error con la marca `"--group-id"` manualmente. • Pruebe a ejecutar el conjunto de pruebas aumentando los tiempos de

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
		El campo `timeout` en los archivos de configuración de la prueba (`test.json`) se ha modificado por error.	espera de prueba. Para obtener más información, consulte Errores de tiempo de espera.
103	PlatformNotSupportError	Se ha definido una combinación incorrecta de sistema operativo y arquitecta en `device.json` .	Cambie la configuración a una de las combinaciones admitidas: • Linux, x86_64 • Linux, ARMv6l • Linux, ARMv7l • Linux, AArch64 • Ubuntu, x86_64 • OpenWRT, ARMv7l • OpenWRT, AArch64 Para obtener más información, consulte Configurar device.json.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
104	VersionNotSupportError	La versión de software de AWS IoT Greengrass Core no es compatible con la versión de IDT que está utilizando.	Utilice el comando device_tester_bin version para buscar la versión compatible del software de AWS IoT Greengrass Core. Por ejemplo, si usa macOS, utilice: ./devicetester_mac_x86_64 version. Para buscar la versión del software de AWS IoT Greengrass Core que está utilizando: • Si ejecuta las pruebas con el software de AWS IoT Greengrass Core preinstalado, utilice SSH para conectarse a su dispositivo principal de AWS IoT Greengrass y ejecute ***<path-to-preinstalled-greengrass-location>*** /greengrass/ggc/core/greengrassd --version

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
			• Si ejecuta las pruebas con una versión distinta del software de AWS IoT Greengrass Core, vaya al directorio `devicetester_greengrass_<os>/products/greengrass/gcc`. La versión del software de AWS IoT Greengrass Core forma parte del nombre del archivo .zip. Puede probar una versión diferente del software de AWS IoT Greengrass Core. Para obtener más información, consulte Empezar con AWS IoT Greengrass.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
105	LanguageNotSupportError	IDT solo admite las bibliotecas y SDK de Python para AWS IoT Greengrass.	Asegúrese de que: • El paquete de SDK bajo `devicetester_greengrass_<os>/products/greengrass/ggsdk` es el SDK de Python. • El contenido de la carpeta `bin` en `devicetester_greengrass_<os>/tests/GGQ_1.0.0/suite/resources/run.runtimefarm/bin` no ha cambiado.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
106	ValidationError	Algunos campos en `device.json` o `config.json` no son válidos.	Compruebe el mensaje de error en la parte derecha del código de error del informe. • Tipo de autenticación no válido para el dispositivo: especifique el método correcto para conectarse al dispositivo. Para obtener más información, consulte the section called “Configurar device.json”. • Invalid private key path: especifique la ruta correcta a la clave privada. Para obtener más información, consulte Configurar device.json. • Región de AWS no válido: especifique un valor de Región de AWS válido en el archivo

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
			`config.json` . Para obtener más información, consulte puntos de enlace de servicio de AWS. • Credenciales de AWS: configure credenciales de AWS válidas en la máquina de pruebas (usando variables de entorno o el archivo `credentials`). Compruebe que el campo `auth` se ha configura do correctam ente. Para obtener más informaci ón, consulte the section called “Cree y configure un Cuenta de AWS”. • Entrada de HSM no válida: compruebe los campos `p11Provid er` , `privateKe yLabel` , `slotLabel` ,

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
			`slotUserPin` y `openSSLEngine` en `device.json`.
107	SSHConnectionFailed	La máquina de pruebas no puede conectarse al dispositivo configurado.	Compruebe que los siguientes campos de su archivo `device.json` son correctos: • `ip` • `user` • `privKeyPath` • `password` Para obtener más información, consulte Configurar device.json.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
108	RunCommandError	Una prueba no ha podido ejecutar un comando en el dispositivo que se está probando.	Compruebe que se permite el acceso raíz al usuario configurado en `device.json` . Algunos dispositivos requieren una contraseña cuando se ejecutan comandos con acceso raíz. Asegúrese de que el acceso raíz se conceda sin contraseña. Para obtener más información, consulte la documentación de su dispositivo. Intente ejecutar manualmente en su dispositivo el comando que ha generado el error y compruebe si también se produce un error.
109	PermissionDeniedError	No hay acceso raíz.	Establezca el acceso raíz para el usuario configurado en el dispositivo.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
110	CreateFileError	No es posible crear un archivo.	Compruebe el espacio en disco del dispositivo y los permisos de directorio.
111	CreateDirError	No se puede crear un directorio.	Compruebe el espacio en disco del dispositivo y los permisos de directorio.
112	InvalidPathError	La ruta al software de AWS IoT Greengrass Core es incorrecta.	Compruebe que la ruta del mensaje de error es válida. No edite ningún archivo del directorio `devicetester_greengrass_` *<os>*.
113	InvalidFileError	Un archivo no es válido.	Compruebe que el archivo en el mensaje de error es válido.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
114	ReadFileError	El archivo especificado no se puede leer.	Compruebe lo siguiente: • Los permisos de los archivos son correctos. • `limits.config` permite abrir un número suficiente de archivos. • El archivo especificado en el mensaje de error existe y es válido. Si está realizando las pruebas en macOS, aumente el límite de archivos abiertos. El límite predeterminado es 256, que es suficiente para las pruebas.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
115	FileNotFoundError	No se ha encontrado un archivo necesario.	Compruebe lo siguiente: • Existe un archivo de Greengrass comprimido en `devicetester_greengrass_<os>/products/greengrass/ggc`. Puede descargar el archivo tar de AWS IoT Greengrass Core desde la página de descargas del Software AWS IoT Greengrass Core.. • El paquete de SDK existe en `devicetester_greengrass_<os>/products/greengrass/ggsdk`. • Los archivos de `devicetester_green`

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
			`grass_ <os>/ tests` no se han modificado.
116	OpenFileFailed	No se puede abrir el archivo especificado.	Compruebe lo siguiente: • El archivo especificado en el mensaje de error existe y es válido. • `limits.config` permite abrir un número suficiente de archivos. Si está realizando las pruebas en macOS, aumente el límite de archivos abiertos. El límite predeterminado es 256, que es suficiente para las pruebas.
117	WriteFileFailed	No se ha podido escribir el archivo (puede ser el DUT o la máquina de pruebas).	Compruebe que existe el directorio especificado en el mensaje de error y que tiene permiso de escritura.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
118	FileCleanUpError	Una prueba no ha podido eliminar el archivo o el directorio especificado, o no pudo desmontar el archivo especificado en el dispositivo remoto.	Si el archivo binario se sigue ejecutando, puede que el archivo esté bloqueado. Detenga el proceso y elimine el archivo especificado.
119	InvalidInputError	Configuración no válida.	Compruebe que el archivo `suite.json` es válido.
120	InvalidCredentialError	Credenciales de AWS no válidas.	• Compruebe sus credenciales de AWS. Para obtener más información, consulte the section called "Configuración de sus credenciales de AWS". • Compruebe la conexión de red y vuelva a ejecutar el grupo de pruebas. Los problemas de red también pueden provocar este error.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
121	AWSSessionError	Error al crear una sesión de AWS.	Este error se produce si las credenciales de AWS no son válidas o si la conexión a Internet es inestable. Intente utilizar la AWS CLI para llamar a una operación de la API de AWS.
122	AWSApiCallError	Se ha producido un error de la API de AWS.	Esto puede deberse a un problema de red. Compruebe su red antes de volver a ejecutar el grupo de pruebas.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
123	IpNotExistError	La dirección IP no está incluida en la información de conectividad.	Compruebe la conexión a Internet. Puede utilizar la consola de AWS IoT Greengrass para comprobar la información de conectividad del objeto de AWS IoT Greengrass principal que se está utilizando en la prueba. Si hay 10 puntos de enlace incluidos en la información de conectividad, puede eliminar algunos o todos ellos y volver a ejecutar la prueba. Para obtener más información, consulte la sección sobre información de conectividad.
124	OTAJobNot CompleteError	No se completó un trabajo de OTA.	Compruebe la conexión a Internet y vuelva a ejecutar el grupo de pruebas de OTA.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
125	CreateGreengrassServiceRoleError	Se ha producido alguno de los siguientes errores: • Se produjo un error al crear un rol. • Se produjo un error al asociar una política al rol de servicio de AWS IoT Greengrass. • La política asociada al rol del servicio no es válida. • Se ha producido un error al asociar un rol a una Cuenta de AWS.	Configure el rol de servicio de AWS IoT Greengrass. Para obtener más información, consulte the section called “Rol de servicio de Greengrass”.
126	DependenciesNotPresentError	Una o varias dependencias necesarias para la prueba específica no están presentes en el dispositivo.	Compruebe el registro de prueba para ver qué dependencias faltan en su dispositivo: *<device-tester-extract-location>* /results/ *<execution-id>*/logs/*<test-case-name.log>*

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
127	InvalidHSMConfiguration	La configuración de HSM/PKCS proporcionada no es correcta.	Proporcione la configuración necesaria para interactuar con el HSM utilizando PKCS#11 en su archivo `device.json` .

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
128	OTAJobNot SuccededError	El trabajo de OTA no se ha podido realizar correctamente.	• Si ejecutó el grupo de pruebas de `ota` individua lmente, ejecute el grupo de pruebas `ggcdepend encies` para comprobar que todas las dependencias (como `wget`) están presentes . A continuación, vuelva a ejecutar el grupo de pruebas de `ota`. • Revise los registros detallados en *`<device-tester-extract-location>`* `/results/` *`<execution-id>`*`/logs/` para obtener informaci ón sobre la solución de problemas y errores. En concreto, compruebe los siguientes registros:

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
			• Registro de consola (`test_manager.log`) • Registro de casos de prueba OTA (`ota_test.log`) • Registro de daemon de GGC (`ota_test_ggc_logs.tar.gz`) • Registros de agentes OTA (`ota_test_ota_logs.tar.gz`) • Compruebe la conexión a Internet y vuelva a ejecutar el grupo de pruebas de `ota`. • Si el problema continúa, póngase en contacto con

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
			AWS Developer Support.
129	NoConnectivityError	Se ha producido un error del agente host al conectar a Internet.	Compruebe la conexión de red y la configuración del firewall. Pruebe de nuevo el grupo de prueba después de que se resuelva el problema de conectividad.
130	NoPermissionError	El usuario de IAM que utiliza para ejecutar IDT para AWS IoT Greengrass no tiene permiso para crear los recursos de AWS necesarios para ejecutar IDT.	Consulte en Plantilla de política de permisos la plantilla que concede los permisos necesarios para ejecutar IDT para AWS IoT Greengrass.

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
131	LeftoverAgentExistError	Su dispositivo está ejecutando procesos de AWS IoT Greengrass cuando intenta iniciar IDT para AWS IoT Greengrass.	Asegúrese de que no haya ningún demonio de Greengrass ejecutándose en su dispositivo. • Puede utilizar este comando para detener el demonio: sudo ./***<absolute-path-to-greengrass-daemon>*** / greengrassd stop. • También puede terminar el demonio de Greengrass de PID. **Note** Si utiliza una instalación existente de AWS IoT Greengrass configurada para iniciarse automáticamente después del reinicio,

Código de error	Nombre del código de error	Causa raíz posible	Solución de problemas
			debe detener el demonio después del reinicio y antes de ejecutar el conjunto de pruebas.
132	DeviceTimeOffsetError	El dispositivo tiene la hora incorrecta.	Ajuste el dispositivo a la hora correcta.
133	InvalidMLConfiguration	La configuración de ML proporcionada es incorrecta.	En el archivo `device.json` , proporcione la configuración correcta necesaria para ejecutar las pruebas de inferencia de ML. Para obtener más información, consulte the section called “Opcional: Configuración del dispositivo para la cualificación ML”.

Resolución de errores de IDT para AWS IoT Greengrass

Cuando se utiliza IDT, debe implantar los archivos de configuración correctos antes de ejecutar IDT para AWS IoT Greengrass. Si obtiene errores de análisis y de configuración, el primer paso es localizar y utilizar una plantilla de configuración adecuada para su entorno.

Si continúa teniendo problemas, consulte el siguiente proceso de depuración.

Temas

¿Dónde puedo buscar los errores?

Los errores de alto nivel se muestran en la consola durante la ejecución y se muestra un resumen de las pruebas fallidas con el error una vez completadas todas las pruebas. `awsiotdevicetester_report.xml` contiene un resumen de todos los errores que han provocado fallos en una prueba. Los archivos de registro de cada ejecución de prueba se almacenan en un directorio determinado con un UUID para la ejecución de pruebas que se muestra en la consola durante la ejecución de la prueba.

El directorio de registros de prueba se encuentra en *<device-tester-extract-location>*`/results/`*<execution-id>*`/logs/`. Este directorio contiene los siguientes archivos que son útiles para la depuración.

Archivos	Descripción
`test_manager.log`	Todos los registros escritos en la consola durante la ejecución de las pruebas. Al final de este archivo se encuentra un resumen de los resultados que incluye una lista de las pruebas con errores.

Archivos	Descripción
	La advertencia y los registros de errores en este archivo pueden proporcionarle información acerca de los errores que se producen.
<test-group-id> __*<test-name>* .log	Registros detallados para la prueba específica.
<test-name> _ggc_logs.tar.gz	Una colección comprimida de todos los registros que AWS IoT Greengrass core daemon ha generado durante la prueba. Para más información, consulte Solución de problemasAWS IoT Greengrass.
<test-name> _ota_logs.tar.gz	Se trata de una colección comprimida de registros generados por el agente OTA de AWS IoT Greengrass durante la prueba. Solo para pruebas OTA.
<test-name> _basic_assertion_publisher_ggad_logs.tar.gz	Colección comprimida de registros generada por el dispositivo del editor de AWS IoT durante la prueba.
<test-name> _basic_assertion_subscriber_ggad_logs.tar.gz	Colección comprimida de registros generados por el dispositivo del suscriptor de AWS IoT durante la prueba.

Errores de procesamiento

Ocasionalmente, un error tipográfico en una configuración de JSON puede dar lugar a errores de análisis. En la mayoría de los casos, el problema es resultado de omitir un paréntesis, una coma o unas comillas en el archivo JSON. IDT realiza la validación JSON e imprime información de depuración. Imprime la línea en la que se produjo el error, el número de línea y el número de la columna del error de sintaxis. Esta información debe ser suficiente para ayudarle a solucionar el error, pero si no logra localizar el error, puede realizar una validación manual en su IDE, en un editor de texto como Atom o Sublime, o a través de una herramienta online como JSONLint.

Error por ausencia de un parámetro obligatorio

Dado que se han añadido nuevas características a IDT, es posible que se hayan introducido cambios en los archivos de configuración. Utilizar un archivo de configuración antiguo podría romper la configuración. Si esto ocurre, el archivo *<test_case_id>*.log en /results/*<execution-id>*/logs enumera explícitamente todos los parámetros que faltan. IDT también valida los esquemas del archivo de configuración JSON para asegurarse de que se ha utilizado la última versión compatible.

Error por la imposibilidad de iniciar una prueba

Es posible que encuentre errores que apuntan a errores durante el inicio de la prueba. Existen varias causas posibles, por lo que debe hacer lo siguiente:

- Asegúrese de que el nombre del grupo que ha incluido en el comando de ejecución existe realmente. Al nombre del grupo se hace referencia directamente desde el archivo device.json.
- Asegúrese de que el dispositivo o dispositivos del grupo tienen parámetros de configuración correctos.

Error por falta de autorización para acceder a un recurso

Es posible que vea el mensaje de error <user or role> is not authorized to access this resource en la salida del terminal o en el archivo test_manager.log en /results/*<execution-id>*/logs. Para resolver este problema, asocie la política administrada AWSIoTDeviceTesterForGreengrassFullAccess al usuario de prueba. Para obtener más información, consulte the section called “Cree y configure un Cuenta de AWS”.

Errores de permiso denegado

IDT realiza operaciones en diversos directorios y archivos en un dispositivo que se está probando. Algunas de estas operaciones requieren acceso raíz. Para automatizar estas operaciones, IDT debe ser capaz de ejecutar comandos con sudo sin escribir una contraseña.

Siga estos pasos para permitir acceso sudo sin escribir una contraseña.

> **Note**
>
> user y username hacen referencia al usuario SSH que utiliza IDT para acceder al dispositivo a prueba.

1. Use sudo usermod -aG sudo ***<ssh-username>*** para añadir el usuario SSH al grupo sudo.
2. Cierre la sesión y, a continuación, vuelva a iniciar sesión para que los cambios surtan efecto.
3. Añada el archivo `/etc/sudoers` y, a continuación, agregue la siguiente línea al final del archivo: *<ssh-username>* `ALL=(ALL) NOPASSWD: ALL`

> **Note**
>
> Le recomendamos que utilice sudo visudo al editar `/etc/sudoers`.

Errores de conexión SSH

Cuando IDT no se puede conectar a un dispositivo a prueba, los errores de conexión se registran en `/results/`*<execution-id>*`/logs/`*<test-case-id>*`.log`. Los mensajes de error de SSH aparecen en la parte superior de este archivo de registro ya que la conexión a un dispositivo bajo prueba es una de las primeras operaciones que realiza IDT.

La mayoría de configuraciones de Windows utilizan la aplicación de terminal PuTTy para conectarse a hosts Linux. Esta aplicación requiere que los archivos de clave privada PEM estándar se conviertan en un formato propio de Windows denominado PPK. Cuando IDT está configurado en su archivo `device.json`, utilice solo archivos PEM. Si utiliza un archivo PPK, IDT no podrá crear una conexión SSH con el dispositivo AWS IoT Greengrass ni tampoco podrá ejecutar pruebas.

Errores de tiempo de espera

Puede aumentar el tiempo de espera de cada prueba especificando un multiplicador de tiempo de espera, que se aplicará al valor predeterminado de cada tiempo de espera de la prueba. Cualquier valor configurado para esta marca debe ser superior o igual a 1,0.

Para utilizar el multiplicador de tiempo de espera, utilice la marca `--timeout-multiplier` al ejecutar las pruebas. Por ejemplo:

```
./devicetester_linux run-suite --suite-id GGQ_1.0.0 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

Para obtener más información, ejecute `run-suite --help`.

Errores No se encuentra el comando al realizar las pruebas

Necesita una versión anterior de la biblioteca de OpenSSL (libssl1.0.0) para ejecutar pruebas en dispositivos de AWS IoT Greengrass. La mayoría de distribuciones de Linux actuales utilizan libssl versión 1.0.2 o posterior (v1.1.0).

Por ejemplo, en una Raspberry Pi, ejecute los siguientes comandos para instalar la versión requerida de libssl:

1. ```
   wget http://ftp.us.debian.org/debian/pool/main/o/openssl/
   libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
   ```
2. ```
   sudo dpkg -i libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
   ```

Excepción de seguridad en macOS

Cuando ejecuta IDT en una máquina host que usa macOS 10.15, el ticket de certificación notarial de IDT no se detecta correctamente y se bloquea la ejecución de IDT. Para ejecutar IDT, tendrá que conceder una excepción de seguridad al `devicetester_mac_x86-64` ejecutable.

Para conceder una excepción de seguridad de IDT ejecutable

1. Abra Preferencias del sistema desde el menú de Apple.
2. Seleccione Seguridad y privacidad, a continuación, en la pestaña General, haga clic en el icono del candado para realizar cambios en la configuración de seguridad.
3. Busque el mensaje `"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.` y seleccione Permitir de todos modos .
4. Acepte la advertencia de seguridad.

Si tiene alguna pregunta acerca de la política de compatibilidad con IDT, póngase en contacto con el servicio de atención al cliente de AWS.

Política de compatibilidad de AWS IoT Device Tester para AWS IoT Greengrass V1

El comprobador de dispositivos (Device Tester, IDT) de AWS IoT para AWS IoT Greengrass es un marco de pruebas descargable que le permite validar y calificar sus dispositivos AWS IoT

Greengrass para su inclusión en el Catálogo de dispositivos de AWS Partner. Es conveniente que utilice la versión más reciente de AWS IoT Greengrass y de IDT para probar o calificar los dispositivos. Para obtener más información, consulte Versiones de IDT admitidas por AWS IoT Greengrass V2 en la Guía del desarrollador de AWS IoT Greengrass Version 2.

También puede utilizar cualquiera de las versiones compatibles de AWS IoT Greengrass e IDT para probar los dispositivos o comprobar si cumplen los requisitos. Aunque puede seguir utilizando versiones no compatibles de IDT, dichas versiones no reciben actualizaciones ni correcciones de errores.

> **Important**
>
> Desde el 4 de abril de 2022, el comprobador de dispositivos (IDT) AWS IoT para AWS IoT Greengrass V1 ya no genera informes de calificación firmados. Ya no puede incluir nuevos dispositivos AWS IoT Greengrass V1 en el Catálogo de dispositivos AWS Partner a través del Programa de calificación de dispositivos AWS. Si bien no puede calificar los dispositivos Greengrass V1, puede seguir usando IDT para AWS IoT Greengrass V1 para probar sus dispositivos Greengrass V1. Le recomendamos que utilice IDT para AWS IoT Greengrass V2 para calificar y publicar los dispositivos Greengrass en el Catálogo de dispositivos AWS Partner.

Si tiene alguna pregunta acerca de la política de compatibilidad, póngase en contacto con el servicio de atención al cliente de AWS.

Solución de problemas de AWS IoT Greengrass

En esta sección, se proporciona información de solución de problemas y posibles soluciones para ayudar a resolver problemas que pueden presentarse en AWS IoT Greengrass.

Para obtener información acerca de las cuotas (límites) de AWS IoT Greengrass, consulte Cuotas de servicio en la Referencia general de Amazon Web Services.

Problemas de AWS IoT Greengrass Core

Si el software de AWS IoT Greengrass Core no se inicia, pruebe los siguientes pasos generales de solución de problemas:

- Asegúrese de instalar los archivos binarios que adecuados para la arquitectura. Para obtener más información, consulte Software AWS IoT Greengrass Core.
- Asegúrese de que el dispositivo del núcleo tiene almacenamiento local disponible. Para obtener más información, consulte the section called “Solución de problemas de almacenamiento”.
- Compruebe `runtime.log` y `crash.log` para ver los mensajes de error. Para obtener más información, consulte the section called “Solución de problemas con los registros”.

Busque en los síntomas y errores siguientes para encontrar información que le ayudará a solucionar problemas con un core AWS IoT Greengrass.

Problemas

- Error: al archivo de configuración le falta el CaPath, CertPath o KeyPath. The Greengrass daemon process with [pid = <pid>] died.
- Error: Failed to parse /<greengrass-root>/config/config.json.
- Error: se ha producido un error al generar la configuración de TLS: ErrUnknown URIScheme
- Error: Runtime failed to start: unable to start workers: container test timed out.
- <address>Error: no se pudo invocar PutLogEvents en un Cloudwatch local, LogGroup:/GreengrassSystem/connection_manager, error:: error al enviar la solicitud debido a: Post http RequestError://<path>/cloudwatch/logs/: dial tcp: getsockopt: conexión rechazada, respuesta: {}.
- Error: Unable to create server due to: failed to load group: chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>: no such file or directory.

- El software de AWS IoT Greengrass Core no se inicia después de cambiar de ejecutarlo sin creación de contenedores a ejecutarlo en un contenedor de Greengrass.
- Error: Spool size should be at least 262144 bytes.
- Error: [ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {"errorString": "operation timed out"}
- Error: container_linux.go:344: starting container process caused "process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"".
- Error: Greengrass daemon running with PID: <id-de-proceso>. Algunos componentes del sistema no se han iniciado. Compruebe si hay errores en ”runtime.log”.
- La sombra del dispositivo no se sincroniza con la nube.
- ERROR: unable to accept TCP connection. accept tcp [::]:8000: accept4: too many open files.
- Error: Runtime execution error: unable to start lambda container. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"".
- Advertencia: [WARN] - [5] GK Remote: error al recuperar los datos de la clave pública: la clave privada no está configurada. ErrPrincipalNotConfigured MqttCertificate
- Error: Permission denied when attempting to use role arn:aws:iam::<account-id>:role/<role-name> to access s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz.
- El núcleo AWS IoT Greengrass está configurado para utilizar un proxy de red y su función de Lambda no puede realizar conexiones salientes.
- El núcleo se encuentra en un bucle infinito de conexión-desconexión. El archivo runtime.log contiene una serie continua de entradas de conexión y desconexión.
- Error: unable to start lambda container. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \\\"proc\\\" to rootfs \\\"
- [ERROR] en tiempo de ejecución: no se puede iniciar el contenedor lambda. {"errorString": "error al inicializar los montajes de contenedor: error al enmascarar la raíz de greengrass en directorio superior superpuesto: error al crear el dispositivo de enmascaramiento en el directorio <ggc-path>: el archivo ya existe"}
- [ERROR]-Deployment failed. {"deploymentId": "<deployment-id>", "errorString": "container test process with pid <pid> failed: container process state: exit status 1"}

- Error: [ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}
- Error: [DEBUG]: no se pudieron obtener rutas. Se va a descartar el mensaje.
- Error: [Errno 24] Too many open <lambda-function>,[Errno 24] Too many open files
- Error: ds server failed to start listening to socket: listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock: bind: invalid argument
- [INFORMACIÓN] (Copiadora) aws.greengrass. StreamManager: stdout. Causado por: com.fasterxml.jackson.databind. JsonMappingException: El instante supera el instante mínimo o máximo
- GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

Error: al archivo de configuración le falta el CaPath, CertPath o KeyPath. The Greengrass daemon process with [pid = <pid>] died.

Solución: Es posible que vea este error en `crash.log` cuando el software de AWS IoT Greengrass Core no se inicia. Esto puede ocurrir si está ejecutando la versión 1.6 o una anterior. Realice una de las acciones siguientes:

- Actualice a versión 1.7 o posterior. Le recomendamos que siempre ejecute la versión más reciente del software de AWS IoT Greengrass Core. Para obtener información sobre descargas, consulte Software AWS IoT Greengrass Core.
- Utilice el formato `config.json` correcto para la versión del software de AWS IoT Greengrass Core. Para obtener más información, consulte the section called “Archivo de configuración de AWS IoT Greengrass Core”.

> **Note**
>
> Para encontrar la versión del software de AWS IoT Greengrass Core instalada en el dispositivo del núcleo, ejecute los siguientes comandos en el terminal del dispositivo.
>
> ```
> cd /greengrass-root/ggc/core/
> sudo ./greengrassd --version
> ```

Error: Failed to parse /<greengrass-root>/config/config.json.

Solución: Es posible que vea este error cuando el software de AWS IoT Greengrass Core no se inicia. Asegúrese de que el archivo de configuración de Greengrass está utilizando un formato de JSON válido.

Abra `config.json` (que se encuentra en `/greengrass-root/config`) y compruebe el formato JSON. Por ejemplo, compruebe que las comas se hayan utilizado correctamente.

Error: se ha producido un error al generar la configuración de TLS: ErrUnknown URIScheme

Solución: Es posible que vea este error cuando el software de AWS IoT Greengrass Core no se inicia. Asegúrese de que las propiedades de la sección crypto del archivo de configuración de Greengrass sean válidas. El mensaje de error debe aportar más información.

Abra `config.json` (que se encuentra en `/greengrass-root/config`) y compruebe la sección `crypto`. Por ejemplo, las rutas de certificados y claves deben utilizar el formato de URI correcto e indicar la ubicación correcta.

Error: Runtime failed to start: unable to start workers: container test timed out.

Solución: Es posible que vea este error cuando el software de AWS IoT Greengrass Core no se inicia. Establezca la propiedad `postStartHealthCheckTimeout` en el archivo de configuración de Greengrass. Esta propiedad opcional configura la cantidad de tiempo (en milisegundos) que el daemon de Greengrass esperará a que finalice la comprobación de estado posterior al inicio. El valor predeterminado es de 30 segundos.

Abra `config.json` (que se encuentra en */greengrass-root*`/config`). En el objeto `runtime`, añada la propiedad `postStartHealthCheckTimeout` y establezca su valor en un número superior a 30000. Añada una coma donde sea necesario para crear un documento JSON válido. Por ejemplo:

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

<address>Error: no se pudo invocar PutLogEvents en un Cloudwatch local, LogGroup:/GreengrassSystem/connection_manager, error:: error al enviar la solicitud debido a: Post http RequestError://<path>/cloudwatch/logs/: dial tcp: getsockopt: conexión rechazada, respuesta: {}.

Solución: Es posible que vea este error cuando el software de AWS IoT Greengrass Core no se inicia. Esto puede ocurrir si está ejecutando AWS IoT Greengrass en un Raspberry Pi y no se ha completado la configuración de memoria necesaria. Para obtener más información, consulte este paso.

Error: Unable to create server due to: failed to load group: chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>: no such file or directory.

Solución: Es posible que vea este error cuando el software de AWS IoT Greengrass Core no se inicia. Si ha implementado un Lambda ejecutable en el núcleo, compruebe la propiedad `Handler` de la función en el archivo `group.json` (localizado en /*greengrass-root*/ggc/deployment/group). Si el controlador no tiene el nombre exacto del ejecutable compilado, reemplace el contenido del archivo `group.json` por un objeto JSON vacío (`{}`) y ejecute los siguientes comandos para iniciar AWS IoT Greengrass:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

A continuación, utilice la API de AWS Lambda para actualizar el parámetro `handler` de la configuración de la función, publique la nueva característica y actualice el alias. Para obtener más información, consulte este artículo sobre el control de versiones y los alias de las funciones de AWS Lambda.

Suponiendo que haya añadido la función a su grupo de Greengrass por alias (recomendado), ahora puede volver a implementar su grupo. (Si no, debe apuntar a la nueva versión o alias de la función en su definición y suscripciones del grupo antes de implementar el grupo).

El software de AWS IoT Greengrass Core no se inicia después de cambiar de ejecutarlo sin creación de contenedores a ejecutarlo en un contenedor de Greengrass.

Solución: Compruebe que no faltan dependencias de contenedor.

Error: Spool size should be at least 262144 bytes.

Solución: Es posible que vea este error cuando el software de AWS IoT Greengrass Core no se inicia. Abra el archivo `group.json` (que se encuentra en /*greengrass-root*/ggc/deployment/

group), sustituya el contenido del archivo por un objeto JSON vacío ({}) y ejecute los comandos siguientes para iniciar AWS IoT Greengrass:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

A continuación, siga los pasos que se indican en el procedimiento the section called “Para almacenar los mensajes en la caché local”. Para la función GGCloudSpooler, asegúrese de especificar un valor GG_CONFIG_MAX_SIZE_BYTES mayor o igual que 262144.

Error: [ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {"errorString": "operation timed out"}

Solución: es posible que vea este error en GGCloudSpooler.log cuando el núcleo de Greengrass no puede enviar mensajes MQTT a AWS IoT Core. Esto puede ocurrir si el entorno de núcleo tiene un ancho de banda limitado y alta latencia. Si está ejecutando AWS IoT Greengrass v1.10.2 o posterior, pruebe a incrementar el valor mqttOperationTimeout en el archivo config.json. Si la propiedad no está presente, agréguela al objeto coreThing. Por ejemplo:

```
{
  "coreThing": {
    "mqttOperationTimeout": 10,
    "caPath": "root-ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    ...
  },
  ...
}
```

El valor predeterminado es 5 y el valor mínimo es 5.

Error: container_linux.go:344: starting container process caused "process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"".

Solución: Es posible que vea este error en `runtime.log` cuando el software de AWS IoT Greengrass Core no se inicia. Esto se produce si `umask` es superior a `0022`. Para resolver este problema, debe establecer `umask` en `0022` o menor. Un valor de `0022` concede a todos los usuarios permiso de lectura a los nuevos archivos de forma predeterminada.

Error: Greengrass daemon running with PID: <id-de-proceso>. Algunos componentes del sistema no se han iniciado. Compruebe si hay errores en ”runtime.log”.

Solución: Es posible que vea este error cuando el software de AWS IoT Greengrass Core no se inicia. Busque información específica de los errores en `runtime.log` y `crash.log`. Para obtener más información, consulte the section called “Solución de problemas con los registros”.

La sombra del dispositivo no se sincroniza con la nube.

Solución: Asegúrese de que AWS IoT Greengrass tiene permisos para las acciones `iot:UpdateThingShadow` y `iot:GetThingShadow` en el rol de servicio de Greengrass. Si el rol de servicio utiliza la política administrada `AWSGreengrassResourceAccessRolePolicy`, estos permisos se incluyen de forma predeterminada.

Consulte Solución de problemas con los tiempos de espera de la sincronización de sombras.

ERROR: unable to accept TCP connection. accept tcp [::]:8000: accept4: too many open files.

Solución: Es posible que vea este error en la salida del script `greengrassd`. Esto puede ocurrir si el límite del descriptor de archivos para el software AWS IoT Greengrass Core ha alcanzado el umbral y debe incrementarse.

Utilice el comando siguiente y reinicie el software de AWS IoT Greengrass Core.

```
ulimit -n 2048
```

> **Note**
>
> En este ejemplo, el límite se aumenta a 2048. Elija un valor adecuado para su caso de uso.

Error: Runtime execution error: unable to start lambda container. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"".

Solución: Instale AWS IoT Greengrass directamente en el directorio raíz o asegúrese de que el directorio donde está instalado el software de AWS IoT Greengrass Core y sus directorios principales tienen permisos `execute` para todo el mundo.

Advertencia: [WARN] - [5] GK Remote: error al recuperar los datos de la clave pública: la clave privada no está configurada. ErrPrincipalNotConfigured MqttCertificate

Solución: AWS IoT Greengrass utiliza un controlador común para validar las propiedades de todas las entidades principales de seguridad. Cabe esperar que se produzca esta advertencia en `runtime.log`, a no ser que se haya especificado una clave privada personalizada para el servidor MQTT local. Para obtener más información, consulte the section called “Entidades de seguridad”.

Error: Permission denied when attempting to use role arn:aws:iam::<account-id>:role/<role-name> to access s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz.

Solución: es posible que aparezca este error cuando se produce un error en una actualización over-the-air (OTA). En la política de rol de firma, añada la Región de AWS de destino como `Resource`. El rol de firmante se utiliza para prefirmar la URL de S3 para la actualización de software de AWS IoT Greengrass. Para obtener más información, consulte Rol de firmante de URL de S3.

El núcleo AWS IoT Greengrass está configurado para utilizar un proxy de red y su función de Lambda no puede realizar conexiones salientes.

Solución: en función del tiempo de ejecución y los ejecutables usados por la función de Lambda para crear las conexiones, es posible que reciba los errores de tiempo de espera de las conexiones. Asegúrese de que sus funciones de Lambda utilicen la configuración de proxy adecuada para conectarse a través del proxy de red. AWS IoT Greengrass pasa la configuración del proxy a las funciones de Lambda definidas por el usuario a través de las variables de entorno `http_proxy`, `https_proxy` y `no_proxy`. Se puede obtener acceso a ellas, tal y como se muestra en el siguiente fragmento de Python.

```
import os
print(os.environ['http_proxy'])
```

Use el mismo formato de mayúsculas/minúsculas que la variable definida en su entorno, por ejemplo, todo en minúsculas `http_proxy` o todo en mayúsculas `HTTP_PROXY`. Para estas variables, AWS IoT Greengrass admite ambos.

> **Note**
>
> La mayoría de las bibliotecas comunes utilizadas para establecer una conexión (como boto3 o cURL y los paquetes `requests` de python) usan estas variables de entorno de forma predeterminada.

El núcleo se encuentra en un bucle infinito de conexión-desconexión. El archivo runtime.log contiene una serie continua de entradas de conexión y desconexión.

Solución: Esto puede ocurrir si otro dispositivo está codificado para utilizar el nombre del objeto del núcleo como ID de cliente para conexiones MQTT a AWS IoT. Las conexiones simultáneas en la misma Región de AWS y Cuenta de AWS deben utilizar ID de cliente únicos. De forma predeterminada, el núcleo utiliza el nombre del objeto del núcleo como ID de cliente para estas conexiones.

Para solucionar este problema, puede cambiar el ID de cliente que utiliza el otro dispositivo para la conexión (recomendado) o anular el valor predeterminado del núcleo.

Para anular el ID de cliente predeterminado del dispositivo del núcleo

1. Ejecute el siguiente comando para detener el daemon de Greengrass.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd stop
   ```

2. Abra *greengrass-root*`/config/config.json` para editarlo como usuario.
3. En el objeto `coreThing`, añada la propiedad `coreClientId` y establezca el valor en su ID de cliente personalizado. El valor debe tener entre 1 y 128 caracteres. Debe ser único en la Región de AWS actual para la Cuenta de AWS.

   ```
   "coreClientId": "MyCustomClientId"
   ```

4. Inicie el daemon.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd start
   ```

Error: unable to start lambda container. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \\\"proc\\\" to rootfs \\\"

Solución: en algunas plataformas, es posible que vea este error en `runtime.log` cuando AWS IoT Greengrass intenta montar el sistema de archivos `/proc` para crear un contenedor de Lambda. O bien, es posible que vea errores similares, como `operation not permitted` o `EPERM`. Estos errores pueden producirse incluso si se superan las pruebas ejecutadas en la plataforma por el script del comprobador de dependencias.

Pruebe una de las siguientes soluciones posibles:

- Habilite la opción `CONFIG_DEVPTS_MULTIPLE_INSTANCES` en el kernel de Linux.
- Establezca las opciones de montaje `/proc` en el host `rw,relatim` únicamente.
- Actualice el kernel de Linux a 4.9 o posterior.

> **Note**
>
> Este problema no está relacionado con el montaje de `/proc` para el acceso a recursos locales.

[ERROR] en tiempo de ejecución: no se puede iniciar el contenedor lambda. {"errorString": "error al inicializar los montajes de contenedor: error al enmascarar la raíz de greengrass en directorio superior superpuesto: error al crear el dispositivo de enmascaramiento en el directorio <ggc-path>: el archivo ya existe"}

Solución: Es posible que vea este error en runtime.log cuando la implementación no se realiza correctamente. Este error se produce si una función de Lambda del grupo AWS IoT Greengrass no puede acceder al directorio `/usr` del sistema de archivos del núcleo.

Para resolverlo, agregue un recurso de volumen local al grupo e implemente el grupo. El recurso debe:

- Especificar `/usr` como ruta de origen y ruta de destino.
- Agregar automáticamente los permisos del SO para el grupo de Linux propietario del recurso.
- Estar afiliado a la función de Lambda y permitir el acceso de solo lectura.

[ERROR]-Deployment failed. {"deploymentId": "<deployment-id>", "errorString": "container test process with pid <pid> failed: container process state: exit status 1"}

Solución: Es posible que vea este error en runtime.log cuando la implementación no se realiza correctamente. Este error se produce si una función de Lambda del grupo AWS IoT Greengrass no puede acceder al directorio `/usr` del sistema de archivos del núcleo.

Puede confirmar que este es el caso comprobando `GGCanary.log` para ver si hay errores adicionales. Si la función de Lambda no puede acceder al directorio `/usr`, `GGCanary.log` contendrá el siguiente error:

```
[ERROR]-standard_init_linux.go:207: exec user process caused "no such file or
 directory"
```

Para resolverlo, agregue un recurso de volumen local al grupo e implemente el grupo. El recurso debe:

- Especificar `/usr` como ruta de origen y ruta de destino.
- Agregar automáticamente los permisos del SO para el grupo de Linux propietario del recurso.
- Estar afiliado a la función de Lambda y permitir el acceso de solo lectura.

Error: [ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}

Solución: Es posible que vea este error en `runtime.log` cuando el software de AWS IoT Greengrass Core no se inicia. Este problema podría ser más común en los sistemas operativos Debian.

Para resolver este problema, siga estos pasos:

1. Actualice el software de AWS IoT Greengrass Core a la versión 1.9.3 o posterior. Esto debería resolver este problema automáticamente.
2. Si sigue apareciendo este error después de actualizar el software AWS IoT Greengrass de Core, defina la propiedad `system.useOverlayWithTmpfs` como `true` en el archivo config.json.

   Example Ejemplo

```
{
  "system": {
    "useOverlayWithTmpfs": true
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

> **Note**
>
> La versión del software de AWS IoT Greengrass Core se muestra en el mensaje de error. Para encontrar la versión del kernel de Linux, ejecute `uname -r`.

Error: [DEBUG]: no se pudieron obtener rutas. Se va a descartar el mensaje.

Solución: compruebe las suscripciones del grupo y asegúrese de que la suscripción que aparece en el mensaje `[DEBUG]` existe.

Error: [Errno 24] Too many open <lambda-function>,[Errno 24] Too many open files

Solución: es posible que vea este error en su archivo de registro de función de Lambda si la función crea instancias `StreamManagerClient` en el controlador de funciones. Le recomendamos que cree el cliente fuera del controlador. Para obtener más información, consulte the section called “Utilizar StreamManagerClient para trabajar con secuencias”.

Error: ds server failed to start listening to socket: listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock: bind: invalid argument

Solución: Es posible que vea este error cuando el software de AWS IoT Greengrass Core no se inicia. Este error se produce cuando el software AWS IoT Greengrass de Core se instala en una carpeta con una ruta de archivo larga. Vuelva a instalar el software AWS IoT Greengrass de Core en una carpeta cuya ruta de archivo tenga menos de 79 bytes, si no utiliza un directorio de escritura, o de 83 bytes, si sí utiliza un directorio de escritura.

[INFORMACIÓN] (Copiadora) aws.greengrass. StreamManager: stdout. Causado por: com.fasterxml.jackson.databind. JsonMappingException: El instante supera el instante mínimo o máximo

Al actualizar el software AWS IoT Greengrass core a la versión 1.11.3, es posible que aparezca el siguiente error en los registros del administrador de secuencias si el administrador de transmisiones no se inicia.

```
2021-07-16T00:54:58.568Z [INFO] (Copier) aws.greengrass.StreamManager:
 stdout. Caused by: com.fasterxml.jackson.databind.JsonMappingException:
 Instant exceeds minimum or maximum instant (through reference chain:
 com.amazonaws.iot.greengrass.streammanager.export.PersistedSuccessExportStatesV1["lastExportTi
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
2021-07-16T00:54:58.579Z [INFO] (Copier) aws.greengrass.StreamManager: stdout.
 Caused by: java.time.DateTimeException: Instant exceeds minimum or maximum instant.
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
```

Si utilizas una versión del software AWS IoT Greengrass core anterior a la 1.11.3 y quieres actualizarla a una versión posterior, utiliza una actualización de OTA para actualizar a la versión 1.11.4.

GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

Si utiliza un dispositivo `apt update` en el que ha instalado el software AWS IoT Greengrass Core desde un repositorio de APT, es posible que aparezca el siguiente error.

```
Err:4 https://dnw9lb6lzp2d8.cloudfront.net stable InRelease
  The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass
 Master Key
Reading package lists... Done
W: GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following
 signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key
```

Este error se produce porque ya AWS IoT Greengrass no ofrece la opción de instalar o actualizar el software AWS IoT Greengrass Core desde el repositorio de APT. Para que se ejecute

correctamente`apt update`, elimine el repositorio AWS IoT Greengrass de la lista de fuentes del dispositivo.

```
sudo rm /etc/apt/sources.list.d/greengrass.list
sudo apt update
```

Problemas de implementación

Utilice la información siguiente como ayuda para solucionar problemas de implementación.

Problemas

- La implementación actual no funciona y desea volver a una implementación funcional anterior.
- Aparece el error 403 Forbidden en la implementación en los registros.
- Se produce un ConcurrentDeployment error al ejecutar el comando create-deployment por primera vez.
- Error: Greengrass is not authorized to assume the Service Role associated with this account, o el error: Failed: TES service role is not associated with this account.
- Error: unable to execute download step in deployment. error while downloading: error while downloading the Group definition file: ... x509: certificate has expired or is not yet valid
- La implementación no finaliza.
- Error: no se pueden encontrar los ejecutables de java o java8, o error: error <deployment-id>de implementación del tipo NewDeployment para el grupo<group-id>: error de trabajo al no <worker-id>poder inicializarse por el motivo La versión de Java instalada debe ser mayor o igual a 8
- La implementación no finaliza y runtime.log contiene varias entradas "wait 1s for container to stop".
- La implementación no finaliza y runtime.log contiene "[ERROR]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"
- <path>Error: error al implementar el <deployment-id>tipo NewDeployment para el grupo<group-id>: error durante el procesamiento. la configuración del grupo no es válida: 112 o [119 0] no tienen el permiso rw en el archivo:.
- Error: < list-of-function-arns > están configurados para ejecutarse como root, pero Greengrass no está configurado para ejecutar funciones de Lambda con permisos de root.

- Error: error en el despliegue <deployment-id>del tipo NewDeployment para el grupo <group-id>Error: error de despliegue de Greengrass: no se pudo ejecutar el paso de descarga en la implementación. Error durante el procesamiento: no se pudo cargar el archivo de grupo descargado: no se pudo encontrar el UID basado en el nombre de usuario, Nombre de usuario: ggc_user: usuario desconocido ggc_user.
- Error: error [ERROR] en tiempo de ejecución: no se puede iniciar el contenedor lambda. {"errorString": "error al inicializar los montajes de contenedor: error al enmascarar la raíz de greengrass en directorio superior superpuesto: error al crear el dispositivo de enmascaramiento en el directorio <ggc-path>: el archivo ya existe"}
- Error: error en <deployment-id>la implementación del tipo NewDeployment para el grupo <group-id>Error: error al iniciar el proceso: container_linux.go:259: al iniciar el proceso contenedor se produjo «process_linux.go:250: al ejecutar exec setns process for init se produjo\" wait: no hay procesos secundarios\ "».
- Error: [WARN]-MQTT[client] dial tcp: lookup <host-prefix>-ats.iot.<region>.amazonaws.com: no existe ete host … [ERROR] -Error de implementación de Greengrass: no se pudo informar del estado de la implementación a la nube... net/http: solicitud cancelada mientras se esperaba la conexión (se superó el tiempo de espera de Client.Timeout mientras se esperaban los encabezados)

La implementación actual no funciona y desea volver a una implementación funcional anterior.

Solución: utilice la consola AWS IoT o la API AWS IoT Greengrass para volver a realizar una implementación de trabajo anterior. Se implementa la versión del grupo correspondiente en su dispositivo principal.

Para volver realizar una implementación (consola), efectúe el siguiente procedimiento:

1. En la página de configuración de grupo, elija Implementar. En esta página se muestra el historial de implementaciones del grupo, incluida la fecha y la hora, la versión del grupo y el estado de cada intento de implementación.
2. Busque la fila que contiene la implementación que desee volver a implementar. Seleccione la implementación que desea volver a implementar y elija Reimplementar.

Para volver a implementar una implementación (CLI), realice el siguiente procedimiento:

1. ListDeploymentsÚselo para buscar el ID de la implementación que desea volver a implementar. Por ejemplo:

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7
```

   El comando devuelve la lista de implementaciones para el grupo.

```
{
    "Deployments": [
        {
            "DeploymentId": "8d179428-f617-4a77-8a0c-3d61fb8446a6",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/8dd1d899-4ac9-4f5d-
afe4-22de086efc62",
            "CreatedAt": "2019-07-01T20:56:49.641Z"
        },
        {
            "DeploymentId": "f8e4c455-8ac4-453a-8252-512dc3e9c596",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/4ad66e5d-3808-446b-940a-
b1a788898382",
            "CreatedAt": "2019-07-01T20:41:47.048Z"
        },
        {
            "DeploymentId": "e4aca044-bbd8-41b4-b697-930ca7c40f3e",
            "DeploymentType": "NewDeployment",
```

```
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/1f3870b6-850e-4c97-8018-
c872e17b235b",
            "CreatedAt": "2019-06-18T15:16:02.965Z"
        }
    ]
}
```

> **Note**
>
> Estos comandos AWS CLI utilizan valores de ejemplo para el grupo y el ID de implementación. Cuando ejecute los comandos, asegúrese de sustituir los valores de ejemplo.

2. Se utiliza CreateDeploymentpara volver a implementar la implementación de destino. Establezca el tipo de implementación en `Redeployment`. Por ejemplo:

```
aws greengrass create-deployment --deployment-type Redeployment \
  --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7 \
  --deployment-id f8e4c455-8ac4-453a-8252-512dc3e9c596
```

   El comando devuelve el ARN y el ID de la nueva implementación.

```
{
    "DeploymentId": "f9ed02b7-c28e-4df6-83b1-e9553ddd0fc2",
    "DeploymentArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/deployments/f9ed02b7-c28e-4df6-83b1-
e9553ddd0fc2"
}
```

3. Se utiliza GetDeploymentStatuspara obtener el estado de la implementación.

Aparece el error 403 Forbidden en la implementación en los registros.

Solución: Asegúrese de que la política del núcleo AWS IoT Greengrass en la nube incluya `"greengrass:*"` como acción permitida.

Se produce un ConcurrentDeployment error al ejecutar el comando create-deployment por primera vez.

Solución: Es posible que haya una implementación en curso. Puede ejecutar get-deployment-status para ver si se ha creado una implementación. En caso contrario, intente volver a crear la implementación.

Error: Greengrass is not authorized to assume the Service Role associated with this account, o el error: Failed: TES service role is not associated with this account.

Solución: Es posible que vea este error cuando la implementación no se realiza correctamente. Compruebe que un rol de servicio de Greengrass esté asociado a su Cuenta de AWS en la Región de AWS actual. Para obtener más información, consulte the section called “Administración del rol de servicio (CLI)” o the section called “Administración del rol de servicio (consola)”.

Error: unable to execute download step in deployment. error while downloading: error while downloading the Group definition file: ... x509: certificate has expired or is not yet valid

Solución: Es posible que vea este error en `runtime.log` cuando la implementación no se realiza correctamente. Si recibe un error `Deployment failed` con el mensaje `x509: certificate has expired or is not yet valid`, compruebe el reloj del dispositivo. Los certificados TLS y X.509 proporcionan una base segura para crear sistemas IoT, pero requieren que los tiempos sean precisos en servidores y clientes. Los dispositivos IoT deben tener la hora correcta (con un margen de 15 minutos) antes de intentar conectarse a AWS IoT Greengrass o a otros servicios de TLS que utilicen certificados de servidor. Para obtener más información, consulte Uso de la hora del dispositivo para validar certificados de servidor AWS IoT en El Internet de las cosas en el blog oficial de AWS.

La implementación no finaliza.

Solución: Haga lo siguiente:

- Asegúrese de que el daemon de AWS IoT Greengrass se está ejecutando en su dispositivo del núcleo. En el terminal de su dispositivo central, ejecute los siguientes comandos para comprobar si el daemon se está ejecutando e inícielo, si es necesario.

  1. Para comprobar si el daemon está en ejecución:

     ```
     ps aux | grep -E 'greengrass.*daemon'
     ```

     Si la salida contiene una entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, el daemon está en ejecución.

     La versión que figura en la ruta depende de la versión del software AWS IoT Greengrass Core que esté instalada en el dispositivo del núcleo.

  2. Iniciar el daemon:

     ```
     cd /greengrass/ggc/core/
     sudo ./greengrassd start
     ```

- Asegúrese de que el dispositivo del núcleo está conectado y los puntos de enlace del núcleo se han configurado correctamente.

Error: no se pueden encontrar los ejecutables de java o java8, o error: error <deployment-id>de implementación del tipo NewDeployment para el grupo<group-id>: error de trabajo al no <worker-id>poder inicializarse por el motivo La versión de Java instalada debe ser mayor o igual a 8

Solución: si el administrador de secuencias está habilitado para el núcleo AWS IoT Greengrass, debe instalar el tiempo de ejecución Java 8 en el dispositivo principal antes de implementar el grupo. Para obtener más información, consulte los requisitos del administrador de secuencias. El administrador de secuencias estará habilitado de forma predeterminada siempre que se utilice el flujo de trabajo de Creación predeterminada de grupos en la consola AWS IoT para crear un grupo.

O deshabilite el administrador de secuencias y, a continuación, implemente el grupo. Para obtener más información, consulte the section called “Configuración (consola)”.

La implementación no finaliza y runtime.log contiene varias entradas "wait 1s for container to stop".

Solución: Ejecute los comandos siguientes en el terminal del dispositivo del núcleo para reiniciar el daemon de AWS IoT Greengrass.

```
cd /greengrass/ggc/core/
sudo ./greengrassd stop
sudo ./greengrassd start
```

La implementación no finaliza y **`runtime.log`** contiene "[ERROR]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"

Solución: puede ver este error en `runtime.log` cuando el núcleo de Greengrass está configurado para usar una conexión proxy HTTPS y la cadena de certificados del servidor proxy no es de confianza en el sistema. Para intentar resolver este problema, agregue la cadena de certificados al certificado de entidad de certificación raíz. El núcleo de Greengrass agrega los certificados desde este archivo al grupo de certificados utilizado para la autenticación TLS en conexiones HTTPS y MQTT con AWS IoT Greengrass.

En el ejemplo siguiente se muestra un certificado de entidad de certificación de servidor proxy agregado al archivo de certificado de entidad de certificación raíz:

```
# My proxy CA
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
```

```
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

De forma predeterminada, el archivo de certificado de entidad de certificación raíz se encuentra en */greengrass-root*`/certs/root.ca.pem`. Para encontrar la ubicación en su dispositivo principal, compruebe la propiedad `crypto.caPath` en config.json.

> **Note**
>
> *greengrass-root* representa la ruta donde está instalado el software de AWS IoT Greengrass Core en su dispositivo. Normalmente, este es el directorio `/greengrass`.

<path>Error: error al implementar el <deployment-id>tipo NewDeployment para el grupo<group-id>: error durante el procesamiento. la configuración del grupo no es válida: 112 o [119 0] no tienen el permiso rw en el archivo:.

Solución: Asegúrese de que el grupo propietario del directorio *<path>* tiene permisos de lectura y escritura en el directorio.

Error: < list-of-function-arns > están configurados para ejecutarse como root, pero Greengrass no está configurado para ejecutar funciones de Lambda con permisos de root.

Solución: Es posible que vea este error en `runtime.log` cuando la implementación no se realiza correctamente. Asegúrese de que ha configurado AWS IoT Greengrass para permitir a las funciones de Lambda funcionar con permisos raíz. Cambie el valor de `allowFunctionsToRunAsRoot` en `greengrass_root/config/config.json` a `yes` o cambie la función de Lambda para que se ejecute como otro usuario/grupo. Para obtener más información, consulte the section called "Ejecución de una función de Lambda como raíz".

Error: error en el despliegue <deployment-id>del tipo NewDeployment para el grupo <group-id>Error: error de despliegue de Greengrass: no se pudo ejecutar el paso de descarga en la implementación. Error durante el procesamiento: no se pudo cargar el archivo de grupo descargado: no se pudo encontrar el UID basado en el nombre de usuario, Nombre de usuario: ggc_user: usuario desconocido ggc_user.

Solución: si la identidad de acceso predeterminada del grupo AWS IoT Greengrass utiliza las cuentas estándar del sistema, el usuario `ggc_user` y el grupo `ggc_group` deben estar presentes en el dispositivo. Para obtener instrucciones que muestran cómo añadir el usuario y el grupo, consulte este paso. Asegúrese de escribir los nombres exactamente tal y como se muestra.

Error: error [ERROR] en tiempo de ejecución: no se puede iniciar el contenedor lambda. {"errorString": "error al inicializar los montajes de contenedor: error al enmascarar la raíz de greengrass en directorio superior superpuesto: error al crear el dispositivo de enmascaramiento en el directorio <ggc-path>: el archivo ya existe"}

Solución: Es posible que vea este error en `runtime.log` cuando la implementación no se realiza correctamente. Este error se produce si una función de Lambda del grupo de Greengrass no puede

acceder al directorio `/usr` del sistema de archivos del núcleo. Para resolverlo, agregue un recurso de volumen local al grupo e implemente el grupo. El recurso debe:

- Especificar `/usr` como ruta de origen y ruta de destino.
- Agregar automáticamente los permisos del SO para el grupo de Linux propietario del recurso.
- Estar afiliado a la función de Lambda y permitir el acceso de solo lectura.

Error: error en <deployment-id>la implementación del tipo NewDeployment para el grupo <group-id>Error: error al iniciar el proceso: container_linux.go:259: al iniciar el proceso contenedor se produjo «process_linux.go:250: al ejecutar exec setns process for init se produjo\" wait: no hay procesos secundarios\ "».

Solución: Es posible que vea este error cuando la implementación no se realiza correctamente. Reintente la implementación.

Error: [WARN]-MQTT[client] dial tcp: lookup <host-prefix>-ats.iot.<region>.amazonaws.com: no existe ete host … [ERROR] -Error de implementación de Greengrass: no se pudo informar del estado de la implementación a la nube... net/http: solicitud cancelada mientras se esperaba la conexión (se superó el tiempo de espera de Client.Timeout mientras se esperaban los encabezados)

Solución: es posible que vea este error si utiliza `systemd-resolved`, que habilita la configuración `DNSSEC` de forma predeterminada. Como resultado, no se reconocen muchos dominios públicos. Los intentos de alcanzar el punto de enlace AWS IoT Greengrass no pueden encontrar el host, por lo que las implementaciones permanecen en el estado `In Progress`.

Puede utilizar los siguientes comandos y resultados para probar este problema. Sustituya el marcador de posición de la *`región`* en los puntos de conexión por su Región de AWS.

```
$ ping greengrass-ats.iot.region.amazonaws.com
ping: greengrass-ats.iot.region.amazonaws.com: Name or service not known
```

```
$ systemd-resolve greengrass-ats.iot.region.amazonaws.com
greengrass-ats.iot.region.amazonaws.com: resolve call failed: DNSSEC validation failed:
 failed-auxiliary
```

Una posible solución consiste en deshabilitar `DNSSEC`. Cuando `DNSSEC` es `false`, las búsquedas de DNS no se validan mediante `DNSSEC`. Para obtener más información, consulte este problema conocido para `systemd`.

1. Agregue `DNSSEC=false` a `/etc/systemd/resolved.conf`.
2. Reinicie `systemd-resolved`.

Para obtener información sobre `resolved.conf` y `DNSSEC`, ejecute `man resolved.conf` en el terminal.

Problemas al crear grupos o funciones

Utilice la siguiente información para ayudar a solucionar problemas con la creación de un grupo AWS IoT Greengrass o función de Lambda de Greengrass.

Problemas

- Error: la configuración '' del grupo no es válida. IsolationMode
- Error: la configuración 'IsolationMode' para la función con arn <function-arn>no es válida.
- Error: MemorySize la configuración de la función con arn <function-arn>no está permitida en IsolationMode =. NoContainer
- Error: la configuración de Access Sysfs para la función con arn <function-arn>no está permitida en =. IsolationMode NoContainer
- Error: la MemorySize configuración de la función con arn <function-arn>es necesaria en =. IsolationMode GreengrassContainer
- Error: la función <function-arn>hace referencia a un recurso de tipo <resource-type>no permitido en IsolationMode =NoContainer.

- Error: Execution configuration for function with arn <arn-de-característica> is not allowed.

Error: la configuración '' del grupo no es válida. IsolationMode

Solución: Este error se produce cuando no se admite el valor `IsolationMode` en `DefaultConfig` de `function-definition-version`. Los valores admitidos son `GreengrassContainer` y `NoContainer`.

Error: la configuración 'IsolationMode' para la función con arn <function-arn>no es válida.

Solución: Este error se produce cuando no se admite el valor `IsolationMode` en <function-arn> de `function-definition-version`. Los valores admitidos son `GreengrassContainer` y `NoContainer`.

Error: MemorySize la configuración de la función con arn <function-arn>no está permitida en IsolationMode =. NoContainer

Solución: este error se produce cuando se especifica un valor `MemorySize` y se decide ejecutar sin contenerización. Las funciones de Lambda que se ejecutan sin creación de contenedores no pueden tener límites de memoria. Puede quitar el límite o cambiar la función de Lambda para ejecutarla en un contenedor de AWS IoT Greengrass.

Error: la configuración de Access Sysfs para la función con arn <function-arn>no está permitida en =. IsolationMode NoContainer

Solución: este error se produce cuando se especifica `true` para `AccessSysfs` y se decide ejecutar sin contenerización. Las funciones de Lambda que se ejecutan sin creación de contenedores deben tener su código actualizado para acceder directamente al sistema de archivos y no pueden usar

`AccessSysfs`. Puede especificar un valor de `false` para `AccessSysfs` o puede cambiar la función de Lambda para ejecutarla en un contenedor de AWS IoT Greengrass.

Error: la MemorySize configuración de la función con arn <function-arn>es necesaria en =. IsolationMode GreengrassContainer

Solución: Este error se produce debido a que no especificó un límite `MemorySize` para una función de Lambda que se está ejecutando en un contenedor de AWS IoT Greengrass. Especifique un valor `MemorySize` para resolver el error.

Error: la función <function-arn>hace referencia a un recurso de tipo <resource-type>no permitido en IsolationMode =NoContainer.

Solución: No puede tener acceso a los tipos de recursos `Local.Device`, `Local.Volume`, `ML_Model.SageMaker.Job`, `ML_Model.S3_Object` o `S3_Object.Generic_Archive` al ejecutar una función de Lambda sin creación de contenedores. Si necesita dichos tipos de recurso, debe ejecutarlos en un contenedor de AWS IoT Greengrass. También puede acceder a dispositivos locales directamente al ejecutar sin creación de contenedores cambiando el código en la función de Lambda.

Error: Execution configuration for function with arn <arn-de-característica> is not allowed.

Solución: Este error se produce al crear una función de Lambda del sistema con `GGIPDetector` o `GGCloudSpooler` y especificar la configuración `IsolationMode` o `RunAs`. Debe omitir los parámetros de `Execution` para esta función de Lambda del sistema.

Problemas de detección

Utilice la siguiente información como ayuda para solucionar problemas con el servicio AWS IoT Greengrass Discovery.

Problemas

Error: el dispositivo es miembro de demasiados grupos, los dispositivos no pueden estar en más de 10 grupos

Solución: se trata de una limitación conocida. Un dispositivo cliente puede ser miembro de hasta 10 grupos.

Problemas con el recurso de machine learning

Utilice la siguiente información para solucionar problemas con los recursos de machine learning.

Problemas

InvalidMLModelOwner : GroupOwnerSetting se proporciona en el recurso del modelo ML, pero GroupOwner o no GroupPermission está presente

Solución: recibe este error si un recurso de aprendizaje automático contiene el ResourceDownloadOwnerSettingobjeto pero la `GroupPermission` propiedad requerida `GroupOwner` o no está definida. Para resolver este problema, defina la propiedad que falta.

NoContainer la función no puede configurar el permiso al adjuntar recursos de Machine Learning. <function-arn>hace referencia a un recurso de aprendizaje automático <resource-id>con permiso <ro/rw> en la política de acceso a los recursos.

Solución: recibirá este error si una función de Lambda que no está en un contenedor especifica permisos de nivel de característica para un recurso de machine learning. Las funciones que no están en un contenedor deben heredar permisos de los permisos de propietario de recursos definidos en el recurso de machine learning. Para resolver este problema, elija heredar permisos de propietario de recursos (consola) o quitar los permisos de la política de acceso a recursos (API) de la función de Lambda.

<function-arn>La función se refiere a un recurso de Machine Learning al que <resource-id>le falta permiso tanto en uno como ResourceAccessPolicy en el recurso OwnerSetting.

Solución: recibirá este error si los permisos para el recurso de machine learning no están configurados para la función de Lambda adjunta o el recurso. Para resolver este problema, configure los permisos en la ResourceAccessPolicypropiedad de la función Lambda o en la OwnerSettingpropiedad del recurso.

<function-arn>La función hace referencia al recurso Machine Learning <resource-id>con el permiso\ "rw\», mientras que la configuración del propietario del recurso GroupPermission solo permite\ "ro\».

Solución: recibirá este error si los permisos de acceso definidos para la función de Lambda adjunta superan los permisos de propietario de recursos definidos para el recurso de machine learning. Para resolver este problema, establezca permisos más restrictivos para la función de Lambda o permisos menos restrictivos para el propietario del recurso.

NoContainer La función <function-arn>hace referencia a los recursos de la ruta de destino anidada.

Solución: recibirá este error si varios recursos de machine learning conectados a una función de Lambda no contenedora utilizan la misma ruta de destino o una ruta de destino anidada. Para resolver este problema, especifique rutas de destino separadas para los recursos.

Lambda <function-arn> obtiene acceso al recurso <resource-id> al compartir el mismo ID de propietario del grupo

Solución: recibirá este error `runtime.log` si se especifica el mismo grupo de sistemas operativos como la identidad Ejecutar como de la función de Lambda y el propietario del recurso de machine learning, pero el recurso no está adjunto a la función de Lambda. Esta configuración da a la función de Lambda permisos implícitos que puede utilizar para acceder al recurso sin autorización de AWS IoT Greengrass.

Para resolver este problema, utilice un grupo de SO diferente para una de las propiedades o adjunte el recurso de machine learning a la función de Lambda.

Problemas del núcleo de AWS IoT Greengrass en Docker

Utilice la información siguiente como ayuda para solucionar problemas relacionados con la ejecución de AWS IoT Greengrass core en un contenedor de Docker.

Problemas

Error: opciones desconocidas: -no-include-email.

Solución: Este error puede producirse al ejecutar el comando `aws ecr get-login`. Asegúrese de que tiene la última versión de la AWS CLI instalada (por ejemplo, ejecute: `pip install awscli --upgrade --user`). Si utiliza Windows e instaló la interfaz de línea de comandos (CLI) mediante el instalador MSI, debe repetir el proceso de instalación. Para obtener más información, consulte Instalación de la AWS Command Line Interface en Microsoft Windows en la Guía del usuario de AWS Command Line Interface.

Advertencia: IPv4 is disabled. Networking will not work.

Solución: Puede que reciba esta advertencia o un mensaje similar al ejecutar AWS IoT Greengrass en un equipo Linux. Habilite el enrutamiento de red IPv4 tal y como se describe en este paso. La implementación en la nube de AWS IoT Greengrass y las comunicaciones MQTT no funcionan cuando el reenvío de IPv4 no está habilitado. Para obtener más información, consulte Configure namespaced kernel parameters (sysctls) at runtime en la documentación de Docker.

Error: A firewall is blocking file Sharing between windows and the containers.

Solución: Puede que reciba este error o un mensaje `Firewall Detected` al ejecutar Docker en un equipo Windows. Esto también puede ocurrir si ha iniciado sesión en una red privada virtual (VPN) y su configuración de red impide el montaje de la unidad compartida. En esta situación, desactive la VPN y vuelva a ejecutar el contenedor Docker.

Error: se produjo un error (AccessDeniedException) al llamar a la GetAuthorizationToken operación: el usuario: arn:aws:iam: ::user/ <account-id><user-name>no está autorizado a realizar: ecr: on resource: * GetAuthorizationToken

Puede recibir este error al ejecutar el comando `aws ecr get-login-password` si no tiene los permisos suficientes para acceder a un repositorio de Amazon ECR. Para obtener más información, consulte los Ejemplos de políticas de repositorios de Amazon ECR y el Acceso a un repositorio de Amazon ECR en la Guía del usuario de Amazon ECR.

Error: Cannot create container for the service greengrass: Conflict. El nombre del contenedor «/» ya está en uso. aws-iot-greengrass

Solución: Esto puede ocurrir cuando el nombre del contenedor se utiliza en un contenedor anterior. Para solucionar este problema, ejecute el siguiente comando para eliminar el contenedor Docker antiguo:

```
docker rm -f $(docker ps -a -q -f "name=aws-iot-greengrass")
```

Error: [FATAL]-Failed to reset thread's mount namespace due to an unexpected error: "operation not permitted". To maintain consistency, GGC will crash and need to be manually restarted.

Solución: Este error de `runtime.log` puede producirse cuando se intenta implementar una función de Lambda `GreengrassContainer` en un AWS IoT Greengrass core que se ejecuta en un contenedor de Docker. Actualmente, solo las funciones de Lambda `NoContainer` pueden desplegarse en un contenedor de Docker de Greengrass.

Para solucionar este problema, asegúrese de que todas las funciones de Lambda se encuentran en modo `NoContainer` e inicie una nueva implementación. A continuación, al iniciar el contenedor, no vincule-monte el directorio `deployment` existente en el contenedor de Docker de AWS IoT Greengrass core. En su lugar, cree un directorio `deployment` vacío y vincúlelo-móntelo en el contenedor de Docker. Esto permite que el nuevo contenedor de Docker reciba la última implementación con funciones de Lambda que se ejecutan en modo `NoContainer`.

Para obtener más información, consulte the section called “Ejecutar AWS IoT Greengrass en un contenedor de Docker”.

Solución de problemas con los registros

Puede configurar los ajustes de registro para un grupo de Greengrass, como enviar registros a Logs, almacenar CloudWatch registros en el sistema de archivos local o ambos. Para obtener información detallada durante la solución de problemas, puede cambiar temporalmente el nivel del registro a `DEBUG`. Los cambios de configuración de registro surten efecto cuando se implementa el grupo. Para obtener más información, consulte the section called “Configuración de registro en AWS IoT Greengrass”.

En el sistema de archivos local, AWS IoT Greengrass almacena registros en las ubicaciones siguientes. La lectura de los registros en el sistema de archivos requiere privilegios raíz.

greengrass-root`/ggc/var/log/crash.log`

Muestra los mensajes generados cuando un AWS IoT Greengrass core se bloquea.

greengrass-root`/ggc/var/log/system/runtime.log`

Muestra mensajes sobre qué componente ha dado error.

greengrass-root/ggc/var/log/system/

Contiene todos los registros de los componentes del sistema de AWS IoT Greengrass, como Certificate Manager y el Connection Manager. Utilizando los mensajes de `ggc/var/log/system/` y `ggc/var/log/system/runtime.log`, debería poder averiguar qué error se produjo en los componentes del sistema de AWS IoT Greengrass.

greengrass-root/ggc/var/log/system/localwatch/

Contiene los registros del AWS IoT Greengrass componente que gestiona la carga de los registros de Greengrass a Logs CloudWatch . Si no puede ver los inicios de sesión de Greengrass CloudWatch, puede utilizarlos para solucionar problemas.

greengrass-root/ggc/var/log/user/

Contiene todos los logs de las funciones de Lambda; definidas por el usuario. Compruebe si existen mensajes de error de las funciones de Lambda locales en esta carpeta.

> **Note**
>
> De forma predeterminada, *greengrass-root* es el directorio `/greengrass`. Si se configura un directorio de escritura, entonces los registros están en ese directorio.

Si los registros están configurados para almacenarse en la nube, utilice los CloudWatch registros para ver los mensajes de registro. `crash.log`solo se encuentra en los registros del sistema de archivos del dispositivo AWS IoT Greengrass principal.

Si AWS IoT está configurado para escribir registros CloudWatch, compruébelos si se producen errores de conexión cuando los componentes del sistema intentan conectarse a ellosAWS IoT.

Para obtener más información acerca del registro en AWS IoT Greengrass, consulte the section called “Monitorización con registros de AWS IoT Greengrass”.

> **Note**
>
> Los registros del software de AWS IoT Greengrass Core v1.0 se almacenan en el directorio *greengrass-root*/var/log.

Solución de problemas de almacenamiento

Cuando el sistema de almacenamiento de archivos local se llene, es posible que algunos componentes empiecen a dar error:

- Las actualizaciones de sombras locales no se realizan.
- Los nuevos certificados de servidor MQTT de AWS IoT Greengrass core no se pueden descargar localmente.
- Las implementaciones producen errores.

Siempre debe conocer la cantidad de espacio libre disponible en el nivel local. Puede calcular el espacio libre en función del tamaño de las funciones de Lambda implementadas, la configuración de la actividad de registro (consulte the section called “Solución de problemas con los registros”) y el número de sombras almacenadas localmente.

Solución de problemas con los mensajes

Todos los mensajes enviados localmente en AWS IoT Greengrass se envían con QoS 0. De forma predeterminada AWS IoT Greengrass almacena los mensajes en un cola en memoria. Por lo tanto, los mensajes sin procesar se pierden cuando se reinicia el núcleo de Greengrass; por ejemplo, después de una implementación de grupo o de un reinicio del dispositivo. Sin embargo, puede configurar AWS IoT Greengrass (versión 1.6 o posterior) para almacenar los mensajes en caché en el sistema de archivos de modo que persistan entre los reinicios del núcleo. También puede configurar el tamaño de la cola. Si configura un tamaño de cola, asegúrese de que sea mayor o igual a 262144 bytes (256 KB). De lo contrario, AWS IoT Greengrass podría no iniciarse correctamente. Para obtener más información, consulte the section called “Cola de mensajes MQTT”.

> **Note**
>
> Cuando se utiliza el valor predeterminado de la cola en memoria, le recomendamos que implemente grupos o reinicie el dispositivo en un momento en el que la interrupción del servicio sea mínima.

También puede configurar el núcleo para establecer sesiones persistentes con AWS IoT. Esto permite que el núcleo reciba mensajes enviados desde la Nube de AWS aunque el núcleo está

desconectado. Para obtener más información, consulte the section called “Sesiones persistentes de MQTT con AWS IoT Core”.

Solución de problemas con los tiempos de espera de la sincronización de sombras

Los retrasos importantes en la comunicación entre un dispositivo del núcleo de Greengrass y la nube podrías provocar un error en la sincronización de sombras debido a que se agote el tiempo de espera. En este caso, debería ver entradas de registro similares a las siguientes:

```
[2017-07-20T10:01:58.006Z][ERROR]-cloud_shadow_client.go:57,Cloud shadow
 client error: unable to get cloud shadow what_the_thing_is_named for
 synchronization. Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
[2017-07-20T10:01:58.006Z][WARN]-sync_manager.go:263,Failed to get cloud
 copy: Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
[2017-07-20T10:01:58.006Z][ERROR]-sync_manager.go:375,Failed to execute sync operation
 {what_the_thing_is_named VersionDiscontinued []}"
```

Una posible solución es configurar la cantidad de tiempo durante la que el dispositivo del núcleo espera un respuesta del host. Abra el archivo config.json de *greengrass-root*`/config` y agregue un campo `system.shadowSyncTimeout` con un valor de tiempo de espera en segundos. Por ejemplo:

```
{
  "system": {
    "shadowSyncTimeout": 10
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

Si no se especifica ningún valor `shadowSyncTimeout` en `config.json`, el valor predeterminado es 5 segundos.

> **Note**
>
> Para el software de AWS IoT Greengrass Core v1.6 y versiones anteriores, el valor predeterminado de `shadowSyncTimeout` es de 1 segundo.

Comprobar AWS re:Post

Si no puede resolver su problema utilizando la información de solución de problemas de este tema, puede buscar en el Solución de problemas o comprobar la etiqueta AWS IoT Greengrass en AWS re:Post para ver si hay problemas relacionados o publicar una nueva pregunta. Los miembros del equipo de AWS IoT Greengrass monitorean activamente AWS re:Post.

Historial de documentos para AWS IoT Greengrass

En la siguiente tabla se describen los cambios importantes en la Guía para AWS IoT Greengrass desarrolladores después de junio de 2018. Para recibir notificaciones sobre los cambios en esta documentación, puede suscribirse a una fuente RSS.

Cambio	Descripción	Fecha
Se actualizó la finalización del soporte para la versión 1.11.x Snap	Se actualizó la información de fin de soporte para la versión 1.11.x de Snap AWS IoT Greengrass Core en snapcraft.io.	22 de septiembre de 2023
Fin del soporte para la versión 1.11.x de Snap	Se agregó la información de fin de soporte para AWS IoT Greengrass core v 1.11.x Snap en snapcraft.io.	19 de septiembre de 2023
Imágenes de Docker para la versión 1.11.6 AWS IoT Greengrass	Las imágenes de Docker para el software AWS IoT Greengrass Core v1.11.6 están disponibles en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub. Le recomendamos que utilice siempre la última versión.	12 de abril de 2022
AWS IoT Device Tester (IDT) por su obsolescencia AWS IoT Greengrass V1	IDT para la AWS IoT Greengrass versión 1 ya no generará informes de calificación firmados.	4 de abril de 2022
Actualización de soporte para AWS IoT Device Tester para AWS IoT Greengrass	IDT para la AWS IoT Greengrass versión 4.4.1 ahora admite el uso de la versión 1.11.6 AWS IoT	24 de marzo de 2022

	Greengrass del software principal para la calificación de los dispositivos.	
AWS IoT Greengrass Publicada la versión 1.11.6	Está disponible la versión 1.11.6 del software AWS IoT Greengrass Core. Estas versiones contienen mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	24 de marzo de 2022
Lanzamiento de la versión 12 SiteWise del conector IoT	Está disponible la versión 12 SiteWise del conector IoT. Esta versión contiene correcciones de errores.	23 de diciembre de 2021
Imágenes de Docker para las versiones AWS IoT Greengras s 1.11.5 y 1.10.5	Las imágenes de Docker para el software AWS IoT Greengrass Core v1.11.5 y v1.10.5 están disponibles en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub. Le recomenda mos que utilice siempre la última versión.	22 de diciembre de 2021
AWS IoT Greengrass V1 política de mantenimiento	La política de AWS IoT Greengrass V1 mantenimi ento define los diferentes niveles de mantenimiento y actualizaciones del AWS IoT Greengrass V1 servicio y del software AWS IoT Greengrass principal, versión 1.x.	20 de diciembre de 2021

AWS IoT Publicada la versión 4.4.1 de Device Tester	Ya está disponible el IDT para AWS IoT Greengras s la versión 4.4.1. Esta versión incluye la versión 1.3.1 del paquete de AWS IoT Greengrass calificación (GGQ) y admite el uso de las versiones AWS IoT Greengras s principales del software, las v1.11.5 y v1.10.5, para la calificación de los dispositivos.	20 de diciembre de 2021
AWS IoT Greengrass publicadas las versiones 1.11.5 y 1.10.5	Están disponibles las versiones 1.11.5 y 1.10.5 del software Core. AWS IoT Greengrass Estas versiones contienen mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	12 de diciembre de 2021

Se han vuelto a publicar las imágenes de Docker de las versiones 1.11.4 y AWS IoT Greengrass 1.10.4	Las imágenes de Docker para las versiones 1.11.4 y 1.10.4 del software AWS IoT Greengrass principal se han vuelto a publicar en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub para corregir errores. BusyBox Para utilizar las imágenes de Docker más recientes, utilice las etiquetas `1.11.4-1` o `1.10.4-1`. Para obtener más información sobre las etiquetas disponibles, consulte amazon/ en Docker Hub. aws-iot-greengrass	8 de diciembre de 2021
CloudWatch El conector Metrics admite marcas de tiempo duplicadas en los datos de entrada	Ahora puede enviar datos de entrada con marcas de tiempo duplicadas a este conector.	19 de noviembre de 2021
Actualización de la prevención del suplente confuso entre servicios	AWS IoT Greengrass admite el uso de las claves de contexto `aws:SourceArn` y de condición `aws:SourceAccount` global en las políticas de recursos de IAM para evitar el confuso problema de los diputados.	1 de noviembre de 2021

Imágenes de Docker para la versión 1.11.4 AWS IoT Greengrass	Las imágenes de Docker para el software AWS IoT Greengrass Core v1.11.4 están disponibles en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub. Le recomendamos que utilice siempre la última versión.	24 de agosto de 2021
Se AWS IoT Greengrass publicó una instantánea de la versión 1.11.4	Está disponible la versión 1.11.4 de la instantánea. AWS IoT Greengrass Le recomendamos que utilice siempre la última versión.	20 de agosto de 2021
Actualización de soporte para AWS IoT Device Tester para AWS IoT Greengrass	IDT para la AWS IoT Greengrass versión 4.1.0 ahora admite el uso de la versión 1.11.4 AWS IoT Greengrass del software principal para la calificación de los dispositivos.	18 de agosto de 2021
AWS IoT Greengrass Publicada la versión 1.11.4	Está disponible la versión 1.11.4 del software AWS IoT Greengrass Core. Esta versión corrige un problema con el administrador de transmisiones que impedía actualizar a la versión 1.11.3 desde una versión anterior del software Core. AWS IoT Greengrass Le recomendamos que utilice siempre la última versión.	17 de agosto de 2021

Puntos de conexión de VPC (AWS PrivateLink)	AWS IoT Greengrass ahora admite puntos finales de VPC de interfaz (AWS PrivateLink) para el AWS IoT Greengras s plano de control. Puede establecer una conexión privada entre su VPC y el plano de control de AWS IoT Greengrass .	16 de agosto de 2021
AWS IoT Publicada la versión 4.1.0 de Device Tester	Ya está disponible la versión 4.1.0 de AWS IoT Device Tester for. AWS IoT Greengrass Esta versión admite el uso de las versiones de software AWS IoT Greengrass principales 1.11.3 y 1.10.4 para la calificación de los dispositivos.	23 de junio de 2021
Se publicó una instantánea de la AWS IoT Greengrass versión 1.11.3	La versión 1.11.3 de la AWS IoT Greengrass instantán ea contiene mejoras de rendimiento y correcciones de errores. Le recomenda mos que siempre se ejecute la versión más reciente.	15 de junio de 2021

Publicadas las imágenes de Docker para las versiones 1.11.3 y AWS IoT Greengrass 1.10.4	Las imágenes de Docker para el software AWS IoT Greengrass Core v1.11.3 y v1.10.4 están disponibles en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub. Estas versiones de Core contienen mejoras de rendimiento y correcciones de errores. AWS IoT Greengrass Le recomendamos que utilice siempre la última versión.	15 de junio de 2021
AWS IoT Greengrass Publicada la versión 1.11.3	Está disponible la versión 1.11.3 del software AWS IoT Greengrass Core. Estas versiones contienen mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	14 de junio de 2021
AWS IoT Greengrass Publicada la versión 1.10.4	Está disponible la versión 1.10.4 del software AWS IoT Greengrass Core. Estas versiones contienen mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	14 de junio de 2021

Se ha lanzado la versión 2 del adaptador de protocolo Modbus-TCP	Está disponible la versión 2 del conector del adaptador de protocolo Modbus-TCP. Esta versión añadió soporte para las cadenas de origen codificadas en ASCII, UTF8 e ISO8859.	24 de mayo de 2021
Lanzamiento de la versión 7 del conector de implement ación de aplicaciones Docker	Está disponible la versión 7 del conector de implement ación de aplicaciones Docker de Greengrass.	5 de abril de 2021
AWS IoT Greengrass Publicada la versión 1.11.1	Está disponible la versión 1.11.1 del software AWS IoT Greengrass Core. Estas versiones contienen mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	29 de marzo de 2021
AWS IoT Publicada la versión 4.0.2 de Device Tester	Está disponible la versión 4.0.2 de AWS IoT Device Tester for. AWS IoT Greengrass Esta versión reemplaza a la versión 4.0.0 de IDT y añade compatibi lidad con la versión 1.11.1 del software Core. AWS IoT Greengrass Esto también corrige un problema que provocaba que IDT ocultara los errores de Hardware Security Integration (HSI).	29 de marzo de 2021

Lanzamiento de la versión 11 SiteWise del conector IoT	Está disponible la versión 11 SiteWise del conector IoT. Esto lanza la compatibilidad con cadenas que contienen caracteres ocultos o no imprimibles. Esta versión también incluye mejoras generales de rendimiento y correcciones de errores.	24 de marzo de 2021
Se volvió a publicar la instantánea de la versión AWS IoT Greengrass 1.11.0	AWS IoT Greengrass La versión 1.11.0 de snap se ha vuelto a publicar en Snapcraft para solucionar errores y un posible bloqueo de la aplicación al utilizar el intérprete de Python. AWS IoT Greengrass no proporciona instantáneas para las versiones de software 1.10 y 1.9.	19 de marzo de 2021
Actualización de soporte para AWS IoT Device Tester para AWS IoT Greengrass	IDT para la AWS IoT Greengrass versión 4.0.0 ahora admite el uso de la versión 1.10.3 AWS IoT Greengrass del software principal para la calificación de los dispositivos.	18 de marzo de 2021
Se AWS IoT Greengrass volvió a publicar la instantánea de la versión 1.8.4	AWS IoT Greengrass La versión 1.8.4 de snap se ha vuelto a publicar en Snapcraft para solucionar errores y un posible bloqueo de la aplicación al utilizar el intérprete de Python.	15 de marzo de 2021

Se volvió a publicar la imagen de Docker de la versión 1.11.0 para AWS IoT Greengrass ARMv7L	La imagen de Docker para la versión 1.11.0 del software AWS IoT Greengrass Core para la plataforma ARMv7L se ha vuelto a publicar en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub para corregir errores y un posible bloqueo de la aplicación al utilizar el intérprete de Python.	8 de marzo de 2021
AWS IoT Greengrass Se publicaron las imágenes de Docker de la versión 1.10.3	Las imágenes de Docker para la versión 1.10.3 del software AWS IoT Greengrass Core ya están disponibles en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub.	8 de marzo de 2021
Se AWS IoT Greengrass han vuelto a publicar las imágenes de Docker de las versiones 1.11.0 y 1.9.4	Las imágenes de Docker para las versiones 1.11.0 y 1.9.4 del software AWS IoT Greengrass Core se han vuelto a publicar en Amazon Elastic Container Registry (Amazon ECR) y Docker Hub para corregir errores y un posible bloqueo de la aplicación al utilizar el intérprete de Python. Las imágenes de Docker para Armv7l no se han vuelto a publicar en este momento.	26 de febrero de 2021

AWS IoT Greengrass Publicada la versión 1.10.3	Está disponible la versión 1.10.3 del software AWS IoT Greengrass Core. Esta versión agrega la propiedad de configuración principal `systemComponentAuthTimeout` y contiene mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	24 de febrero de 2021
Lanzamiento de la versión 10 SiteWise del conector IoT	Está disponible la versión 10 SiteWise del conector IoT. Esta versión resuelve los problemas de estabilidad del StreamManager cliente cuando se pierde la conexión y mejora el manejo de los valores del OPC-UA cuando `SourceTimestamp` no existe a. Utilice el SiteWise conector de IoT para enviar datos de dispositivos y equipos locales a las propiedades de los activos en IoT SiteWise.	22 de enero de 2021

Lanzamiento de la versión 9 SiteWise del conector IoT	Está disponible la versión 9 SiteWise del conector IoT. Esto lanza la compatibilidad con los destinos de transmisión `StreamManager` personalizados de Greengrass, la banda muerta OPC-UA, el modo de escaneo personalizado y la velocidad de escaneo personalizada. Esto también incluye un rendimiento mejorado durante las actualizaciones de configuración realizadas desde la SiteWise puerta de enlace de IoT. Utilice el SiteWise conector de IoT para enviar datos de dispositivos y equipos locales a las propiedades de los activos en IoT SiteWise.	15 de diciembre de 2020
AWS IoT Publicada la versión 4.0.0 de Device Tester	Está disponible la versión 4.0.0 de AWS IoT Device Tester for. AWS IoT Greengrass Esta versión le permite usar IDT para desarrollar y ejecutar sus conjuntos de pruebas personalizadas para la validación de dispositivos. Esto también incluye aplicaciones IDT firmadas con código para macOS y Windows.	15 de diciembre de 2020

AWS IoT Greengrass snap v1.11	La versión 1.11.0 del AWS IoT Greengrass snap admite funciones Lambda no contenedorizadas. Le recomendamos que siempre se ejecute la versión más reciente.	6 de diciembre de 2020
Lanzamiento de la versión 8 SiteWise del conector IoT	Está disponible la versión 8 SiteWise del conector IoT. Esta versión mejora la estabilidad cuando el conector experimenta una conectividad de red intermitente. Utilice el SiteWise conector de IoT para enviar datos de dispositivos y equipos locales a las propiedades de los activos en IoT SiteWise.	19 de noviembre de 2020
El conector Kinesis Firehose admite el modo sin contenedor	Puede usar el parámetro `IsolationMode` para configurar el modo de creación de contenedores del conector.	19 de octubre de 2020
Lanzamiento de la versión 6 del conector de implementación de aplicaciones Docker	Está disponible la versión 6 del conector de implementación de aplicaciones Docker de Greengrass.	18 de septiembre de 2020

AWS IoT Greengrass Publicada la versión 1.11.0	Está disponible la versión 1.11.0 del software AWS IoT Greengrass Core. Esta versión añade la caracterí stica de telemetría del estado del sistema y una API de comprobación de estado local. Stream Manager ahora puede exportar datos a Amazon Simple Storage Service (Amazon S3) e IoT. SiteWise Esta versión también contiene mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	16 de septiembre de 2020
Lanzamiento de la versión 7 SiteWise del conector IoT	Está disponible la versión 7 SiteWise del conector IoT. Esta versión corrige un problema con las métricas de puerta de enlace. Utilice el SiteWise conector de IoT para enviar datos de dispositi vos y equipos locales a las propiedades de los activos en IoT SiteWise.	14 de agosto de 2020
ServiceNow MetricBase Los conectores Integration, Splunk Integration y Twilio Notificat ions admiten el modo sin contenedor	Puede usar el parámetro `IsolationMode` para configurar el modo de creación de contenedores del conector.	30 de julio de 2020

El conector SNS admite el modo sin contenedor	Puede usar el parámetro `IsolationMode` para configurar el modo de creación de contenedores del conector.	6 de julio de 2020
CloudWatch El conector Metrics admite el modo sin contenedor	Puede usar el parámetro `IsolationMode` para configurar el modo de creación de contenedores del conector.	17 de junio de 2020
AWS IoT Greengrass Publicada la versión 1.10.2	Está disponible la versión 1.10.2 del software AWS IoT Greengrass Core. Esta versión agrega la propiedad de configuración principal `mqttOperationTimeout` y contiene mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	8 de junio de 2020
Los instaladores de machine learning de Tensorflow se han quedado obsoletos	AWS IoT Greengrass Los instaladores de aprendizaje automático preempaquetados de Tensorflow han quedado obsoletos. Los ejemplos de aprendizaje automático se han actualizado a Python 3.7.	29 mayo de 2020

Compatibilidad con el marco Chainer y los instaladores de machine learning de Greengrass obsoletos	AWS IoT Greengrass Los instaladores y descargas de aprendizaje automático preempaquetados para MXNet y DLR han quedado obsoletos . El marco Chainer admite las descargas asociadas que se han quedado obsoletas.	4 de mayo de 2020
Lanzamiento de la versión 6 SiteWise del conector IoT	Está disponible la versión 6 SiteWise del conector IoT. Esta versión añade compatibi lidad con las CloudWatc h métricas y la detección automática de nuevas etiquetas OPC-UA. Esto significa que no es necesario reiniciar la gateway cuando las etiquetas cambien en las fuentes de OPC-UA. Esta versión del conector requiere el administrador de transmisiones y el software AWS IoT Greengrass Core v1.10.0 o superior. Utilice el SiteWise conector de IoT para enviar datos de dispositi vos y equipos locales a las propiedades de los activos en IoT SiteWise.	29 de abril de 2020

Conectores actualizados a Python 3.7	Los conectores compatibles con el entorno de ejecución de Python se han actualizado a Python 3.7. Le recomendamos que actualice las versiones de los conectores de Python 2.7 a Python 3.7.	29 de abril de 2020
La configuración del dispositivo Greengrass puede ejecutarse en modo silencioso	Puede ejecutar la configuración del dispositivo Greengrass en modo silencioso para que el script no le pida ningún valor.	27 de abril de 2020
Nuevas imágenes base de Docker	Puede descargar imágenes de AWS IoT Greengrass Docker basadas en imágenes base de Alpine Linux (x86_64, ARMv7L o AArch64).	23 de abril de 2020
AWS IoT Greengrass Publicada la versión 1.10.1	Está disponible la versión 1.10.1 del software AWS IoT Greengrass Core. Estas versiones contienen mejoras de rendimiento y correcciones de errores. Le recomendamos que utilice siempre la última versión.	16 de abril de 2020
Nuevo capítulo de seguridad	AWS IoT Greengrass el contenido de seguridad se ha reorganizado y se ha agregado nueva información.	30 de marzo de 2020

Utilice el administrador de paquetes APT para instalar AWS IoT Greengrass	En las distribuciones de Linux compatibles basadas en Debian, puede utilizarlas `apt` para instalar el software AWS IoT Greengrass Core en sus dispositivos.	26 de febrero de 2020
Lanzamiento de la versión 5 SiteWise del conector IoT	Está disponible la versión 5 SiteWise del conector IoT. Esta versión corrige un problema de compatibilidad con la versión AWS IoT Greengrass 1.9.4 del software Core. Utilice el SiteWise conector de IoT para enviar datos de dispositivos y equipos locales a las propiedades de los activos en IoT SiteWise.	12 de febrero de 2020
Nuevo script para configurar rápidamente un dispositivo central	Puede usar la configuración del dispositivo Greengrass para configurar el dispositivo principal en solo unos minutos. Además, AWS IoT Greengrass ahora es compatible con las funciones Lambda 12.x de Node.js.	20 de diciembre de 2019

AWS IoT Greengrass Publicada la versión 1.10.0	Está disponible la versión 1.10.0 del software AWS IoT Greengrass Core. Las nuevas características de esta versión incluyen un administrador de flujos, soporte de contenedores con el conector de implement ación de aplicaciones Docker, funciones de Lambda que no están en un contenedor y que pueden acceder a recursos de machine learning, soporte para sesiones MQTT persisten tes con AWS IoT y el tráfico MQTT local que puede viajar a través de un puerto específico.	25 de noviembre de 2019
Compatibilidad de la consola con las notificaciones de implementación	Usa la EventBridge consola de Amazon para crear reglas de eventos que se activen cuando las implementaciones de tu grupo de Greengrass cambien de estado.	14 de noviembre de 2019
AWS IoT Greengrass Publicada la versión 1.9.4	Está disponible la versión 1.9.4 del software AWS IoT Greengrass Core. Estas versiones contienen mejoras de rendimiento y correccio nes de errores. Como práctica recomendada, le recomenda mos que siempre se ejecute la versión más reciente.	17 de octubre de 2019

Compatibilidad de la consola para administrar el rol de servicio de Greengrass	Utilice las funciones nuevas y mejoradas de la AWS IoT consola para gestionar su función de servicio de Greengrass.	4 de octubre de 2019
Compatibilidad de la consola para administrar etiquetas en el nivel de grupo	Puede crear, ver y administrar etiquetas para sus grupos de Greengrass en la consola de AWS IoT .	23 de septiembre de 2019
Nuevos conectores de machine learning	Utilice el conector de comentarios de aprendiza je automático para publicar la entrada y las prediccio nes del modelo y el conector de detección de objetos de aprendizaje automático para ejecutar un servicio de inferencia de detección de objetos local.	19 de septiembre de 2019
AWS IoT Greengrass Publicada la versión 1.9.3	Está disponible la versión 1.9.3 del software AWS IoT Greengrass Core. Esta versión le permite instalar el software AWS IoT Greengrass Core en distribuciones de Raspbian en arquitecturas ARMv6L, admite actualizaciones OTA en el puerto 443 con ALPN y contiene una corrección de errores para las cargas binarias enviadas desde las funciones Lambda de Python 2.7 a otras funciones de Lambda.	12 de septiembre de 2019

AWS IoT Greengrass Publicada la versión 1.8.4	Está disponible la versión 1.8.4 del software AWS IoT Greengrass Core. Estas versiones contienen mejoras de rendimiento y correcciones de errores. Si está ejecutando la versión 1.8.x, le recomendamos que actualice a la versión 1.8.4 o 1.9.3. Para versiones anteriores, le recomendamos que actualice a la versión 1.9.3.	30 de agosto de 2019
AWS IoT Greengrass se lanzó la versión 1.9.2 con soporte para OpenWrt	Está disponible la versión 1.9.2 del software AWS IoT Greengrass Core. Esta versión le permite instalar el software AWS IoT Greengrass Core en OpenWrt distribuciones con arquitecturas Armv8 (AArch64) y ARMv7L.	20 de junio de 2019
AWS IoT Greengrass Publicada la versión 1.8.3	Está disponible la versión 1.8.3 del software AWS IoT Greengrass Core. Esta versión contiene mejoras de rendimiento generales y correcciones de errores. Si está ejecutando la versión 1.8.x, le recomendamos que actualice a la versión 1.8.3 o 1.9.2. Para versiones anteriores, le recomendamos que actualice a la versión 1.9.2.	20 de junio de 2019

AWS IoT Greengrass Publicada la versión 1.9.1	Está disponible la versión 1.9.1 del software AWS IoT Greengrass Core. Esta versión contiene una corrección de errores para los mensajes AWS IoT que contienen un carácter comodín en el tema.	10 de mayo de 2019
AWS IoT Greengrass Publicada la versión 1.8.2	Está disponible la versión 1.8.2 del software AWS IoT Greengrass Core. Esta versión contiene mejoras de rendimiento generales y correcciones de errores. Si está ejecutando la versión 1.8.x, le recomendamos que actualice a la versión 1.8.2 o 1.9.0. Para versiones anteriores, le recomendamos que actualice a la versión 1.9.0.	2 de mayo de 2019
AWS IoT Greengrass Publicada la versión 1.9.0	Nuevas características: compatibilidad con los tiempos de ejecución de Lambda para Python 3.7 y Node.js 8.10, conexiones MQTT optimizadas y compatibilidad con claves de Elliptic Curve (EC) para el servidor MQTT local.	1 de mayo de 2019

AWS IoT Greengrass publicada la versión 1.8.1	Está disponible la versión 1.8.1 del software AWS IoT Greengrass Core. Esta versión contiene mejoras de rendimiento generales y correcciones de errores. Como práctica recomendada, le recomendamos que siempre se ejecute la versión más reciente.	18 de abril de 2019
AWS IoT Greengrass snap disponible en snapcraft	Usa la aplicación AWS IoT Greengrass Snap Store para diseñar, probar e implementar software rápidamente en dispositivos Linux con AWS IoT Greengrass.	1 de abril de 2019
Compatibilidad con mayores controles de acceso utilizando permisos basados en etiquetas	Puedes usar etiquetas en las políticas AWS Identity and Access Management (IAM) para controlar el acceso a tus AWS IoT Greengrass recursos.	29 de marzo de 2019
Lanzamiento del conector IoT Analytics	Utilice el conector IoT Analytics para enviar los datos del dispositivo local a canales de AWS IoT Analytics .	15 de marzo de 2019
Soporte por lotes en el conector Kinesis Firehose	El conector Kinesis Firehose permite enviar registros de datos por lotes a Amazon Data Firehose en un intervalo específico.	15 de marzo de 2019

AWS CloudFormationAWS IoT Greengrass soporte para recursos	Use AWS CloudFormation plantillas para crear y administrar AWS IoT Greengrass recursos.	15 de marzo de 2019
AWS IoT Greengrass Publicada la versión 1.8.0	Nuevas funciones: identidad de acceso predeterminada configurable para las funciones de Lambda, compatibilidad con el tráfico HTTPS a través del puerto 443 e identificadores de cliente con nombres predecibles para las conexiones MQTT con. AWS IoT	7 de marzo de 2019
AWS IoT Greengrass Publicadas las versiones 1.7.1 y 1.6.1	Están disponibles las versiones 1.7.1 y 1.6.1 del software AWS IoT Greengrass Core. Estas versiones requieren el kernel de Linux versión 3.17 o posterior. Recomendamos a los clientes que ejecutan cualquier versión del software de núcleo de Greengrass que actualicen a la versión 1.7.1 inmediatamente.	11 de febrero de 2019
SageMaker Neo Deep Learning Runtime	El tiempo de ejecución de aprendizaje profundo de SageMaker Neo es compatible con los modelos de aprendizaje automático que han sido optimizados por el compilador de aprendizaje profundo de SageMaker Neo.	28 de noviembre de 2018

Se ejecuta AWS IoT Greengrass en un contenedor de Docker	Puede ejecutar AWS IoT Greengrass en un contenedo r de Docker configurando su grupo de Greengrass para que se ejecute sin contenerización.	26 de noviembre de 2018
AWS IoT Greengrass Publicada la versión 1.7.0	Nuevas características: conectores de Greengras s, Secrets Manager local, aislamiento y configuración de permisos para funciones de Lambda, raíz de hardware de seguridad de confianza , conexión mediante ALPN o proxy de red y soporte de Raspbian Stretch.	26 de noviembre de 2018
AWS IoT Greengrass descargas de software	Los paquetes de software AWS IoT Greengrass Core, AWS IoT Greengrass Core SDK y AWS IoT Greengras s Machine Learning SDK están disponibles para su descarga a través de Amazon CloudFront.	26 de noviembre de 2018
AWS IoT Device Tester para AWS IoT Greengrass	Utilice AWS IoT Device Tester AWS IoT Greengrass para comprobar que la arquitect ura de la CPU, la configura ción del núcleo y los controlad ores funcionan correctamente. AWS IoT Greengrass	26 de noviembre de 2018

AWS CloudTrail registro de llamadas a la AWS IoT Greengrass API	AWS IoT Greengrass está integrado con AWS CloudTrail un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un AWS servicio en AWS IoT Greengrass.	29 de octubre de 2018
Support para la TensorFlow versión 1.10.1 en NVIDIA Jetson TX2	La biblioteca TensorFlow precompilada para NVIDIA Jetson TX2 que se proporciona ahora usa la versión 1.10.1. AWS IoT Greengrass TensorFlow Esto permite utilizar Jetpack 3.3 y CUDA Toolkit 9.0.	18 de octubre de 2018
Compatibilidad con los recursos de machine learning de MXNet versión 1.2.1	AWS IoT Greengrass admite modelos de aprendizaje automático entrenados con MXNet v1.2.1.	29 de agosto de 2018
AWS IoT Greengrass publicada la versión 1.6.0	Nuevas características: ejecutables de Lambda, cola de mensajes configurable, intervalo de reintento de reconexión configurable, recursos de volumen bajo / proc y directorio de escritura configurable.	26 de julio de 2018

Actualizaciones anteriores

En la siguiente tabla se describen los cambios importantes en la Guía para AWS IoT Greengrass desarrolladores antes de julio de 2018.

Cambio	Descripción	Fecha
AWS IoT Greengrass Publicada la versión 1.5.0	Nuevas características: • Inferencia del aprendizaje automático local mediante modelos entrenados en la nube. Para obtener más información, consulte Cómo realizar la inferencia de machine learning. • Las funciones de Lambda de Greengrass admiten datos de entrada binarios, además de JSON. Para obtener más información, consulte Precios de AWS IoT Greengrass Core.	29 de marzo de 2018
AWS IoT Greengrass Publicada la versión 1.3.0	Nuevas características: • O ver-the-air (OTA) agente de actualización capaz de gestionar trabajos de actualización de Greengrass desplegados en la nube. Para obtener más información, consulte Actualizaciones de OTA para el software AWS IoT Greengrass Core. • Acceso a periféricos y recursos locales desde las funciones de Lambda de Greengrass. Para obtener más información, consulte Acceder a recursos locales con conectores y funciones de Lambda.	27 de noviembre de 2017
AWS IoT Greengrass Publicada la versión 1.1.0	Nuevas características: • Restablecer los AWS IoT Greengrass grupos desplegados. Para obtener más información, consulte Restablecimiento de implementaciones. • Compatibilidad con los tiempos de ejecución de Lambda de Node.js 6.10 y Java 8, además de Python 2.7.	20 de septiembre de 2017
AWS IoT Greengrass Publicada la versión 1.0.0	AWS IoT Greengrass está disponible de forma general.	7 de junio de 2017