aws

Guía del usuario

FreeRTOS

FreeRTOS: Guía del usuario

Table of Contents

¿Qué es FreeRTOS?

Desarrollado en colaboración con las principales compañías de chips del mundo durante un período de 15 años y ahora descargado cada 170 segundos, FreeRTOS es un sistema operativo en tiempo real (RTOS) líder del mercado para microcontroladores y microprocesadores. Distribuido libremente bajo la licencia de código abierto del MIT, FreeRTOS incluye un kernel y un conjunto creciente de bibliotecas apropiadas para su uso en todos los sectores de la industria. FreeRTOS se basa en la fiabilidad y la facilidad de uso.

Freertos incluye bibliotecas de conectividad, seguridad y actualizaciones over-the-air (OTA). FreeRTOS también incluye aplicaciones de demostración que muestran las características de FreeRTOS en placas calificadas.

FreeRTOS es un proyecto de código abierto. Puede descargar el código fuente, contribuir con cambios o mejoras o informar de problemas en el GitHub sitio en https://github.com/FreeRTOS/FreeRTOS.

Hemos lanzado el código de FreeRTOS con la licencia de código abierto MIT, por lo que puede utilizarlo en proyectos comerciales y personales.

También agradecemos las contribuciones a la documentación de FreeRTOS (Guía del usuario de FreeRTOS, Guía de portabilidad de FreeRTOS y Guía de calificación de FreeRTOS). Para ver la fuente de Markdown de la documentación, consulta https://github.com/awsdocs/ aws-freertos-docs. Se publica con la licencia Creative Commons (CC BY-ND).

Descarga del código fuente de FreeRTOS

Descargue los paquetes más recientes de FreeRTOS y soporte a largo plazo (LTS) desde la página de descargas de freertos.org.

Control de versiones de FreeRTOS

Las bibliotecas individuales utilizan números de versión de estilo x.y.z, de forma similar al control de versiones semántico. x es el número de versión principal, y es el número de versión secundario y, a partir de 2022, z es un número de parche. Antes de 2022, z era un número de publicación puntual, por lo que las primeras bibliotecas LTS debían tener un número de parche con el formato “x.y.z Parche LTS 2”.

Los paquetes de la biblioteca utilizan números de versión con registro de fecha con el estilo aaaamm.x. aaaa es el año, mm el mes y x un número de secuencia opcional que muestra el orden de publicación dentro del mes. En el caso del paquete LTS, x es un número de parche secuencial para esa versión de LTS. Las bibliotecas individuales contenidas en un paquete son cualquiera que sea la última versión de esa biblioteca en esa fecha. En el caso del paquete LTS, es la última versión de parche de las bibliotecas LTS que se publicó originalmente como versión LTS en esa fecha.

Soporte a largo plazo de FreeRTOS

Las versiones de soporte a largo plazo (LTS) de FreeRTOS reciben correcciones de seguridad y errores críticos (si fuera necesario) durante al menos dos años después de su publicación. Con este mantenimiento continuo, puede incorporar correcciones de errores a lo largo de un ciclo de desarrollo e implementación sin la costosa interrupción de actualizar a las nuevas versiones principales de las bibliotecas de FreeRTOS.

Con LTS de FreeRTOS, obtiene el conjunto completo de bibliotecas necesarias para crear productos integrados y de IoT conectados y seguros. LTS ayuda a reducir los costes de mantenimiento y pruebas asociados a la actualización de las bibliotecas de los dispositivos que ya están en producción.

LTS de FreeRTOS incluye el kernel de FreeRTOS y las bibliotecas de IoT: FreeRTOS+TCP, coreMQTT, coreHTTP, corePKCS11, coreJSON, OTA de AWS IoT, trabajos de AWS IoT y sombra de dispositivo de AWS IoT Device Defender. Para obtener más información, consulte las bibliotecas LTS de FreeRTOS.

Plan de mantenimiento ampliado de FreeRTOS

AWS también ofrece el plan de mantenimiento ampliado (EMP) de FreeRTOS, que proporciona parches de seguridad y correcciones de errores críticos en la versión de soporte a largo plazo (LTS) de FreeRTOS elegida durante un máximo de diez años adicionales. Con el EMP de FreeRTOS, sus dispositivos de larga duración basados en FreeRTOS pueden confiar en una versión con estabilidad de características y que recibe actualizaciones de seguridad durante años. Recibe notificaciones puntuales de los próximos parches en las bibliotecas de FreeRTOS, para que pueda planificar la implementación de parches de seguridad en sus dispositivos de Internet de las cosas (IoT).

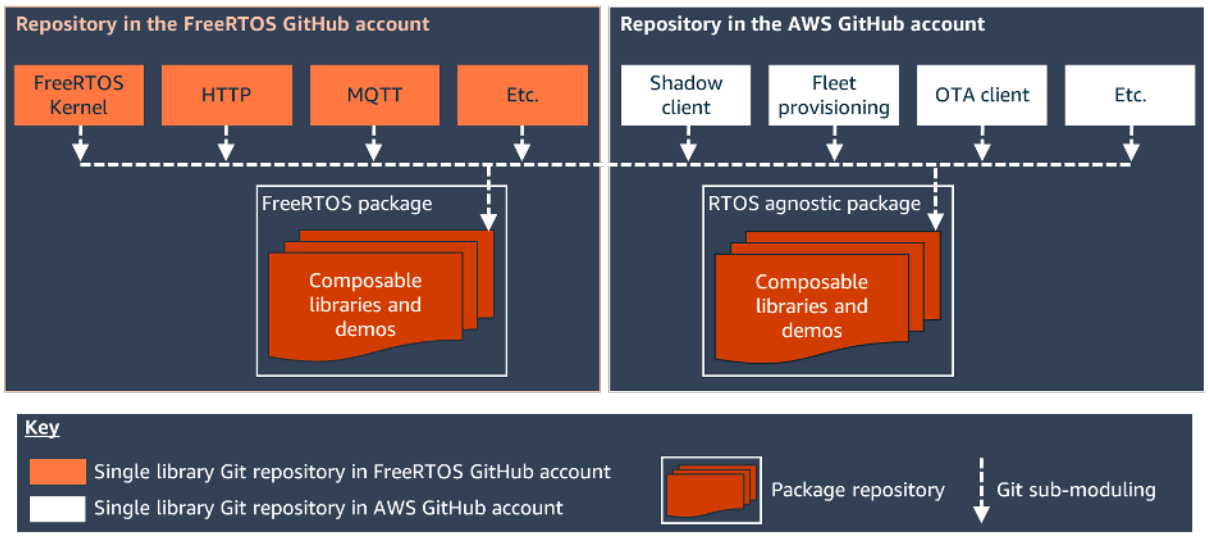
Para obtener más información sobre EMP de FreeRTOS, consulte la página Características.

Arquitectura de FreeRTOS

FreeRTOS contiene dos tipos de repositorios, repositorios de bibliotecas únicas y repositorios de paquetes. Cada repositorio de bibliotecas únicas contiene el código fuente de una biblioteca sin ningún proyecto de compilación ni ejemplos. Los repositorios de paquetes contienen varias bibliotecas y pueden contener proyectos preconfigurados que demuestren el uso de la biblioteca.

Si bien los repositorios de paquetes contienen varias bibliotecas, no contienen copias de esas bibliotecas. En cambio, los repositorios de paquetes hacen referencia a las bibliotecas que contienen como submódulos de git. El uso de submódulos garantiza que haya una única fuente de información fiable para cada biblioteca individual.

Los repositorios de git de las bibliotecas individuales se dividen en dos GitHub organizaciones. Los repositorios que contienen bibliotecas específicas de FreeRTOS (como FreeRTOS+TCP) o bibliotecas genéricas (como CoreMQTT, que no depende de la nube porque funciona con cualquier broker de MQTT) pertenecen a la organización FreeRTOS. GitHub Los repositorios que contienen bibliotecas AWS IoT específicas (como el cliente de actualización) se encuentran en la organización. AWS IoT over-the-air AWS GitHub En el siguiente diagrama se explica la estructura.



Plataformas de hardware calificadas de FreeRTOS

Las siguientes plataformas de hardware están calificadas para FreeRTOS:

- Kit de aprovisionamiento automático ATECC608A para AWS IoT
- Kit de desarrollo Cypress CYW943907AEVAL1F
- Kit de desarrollo Cypress CYW954907AEVAL1F

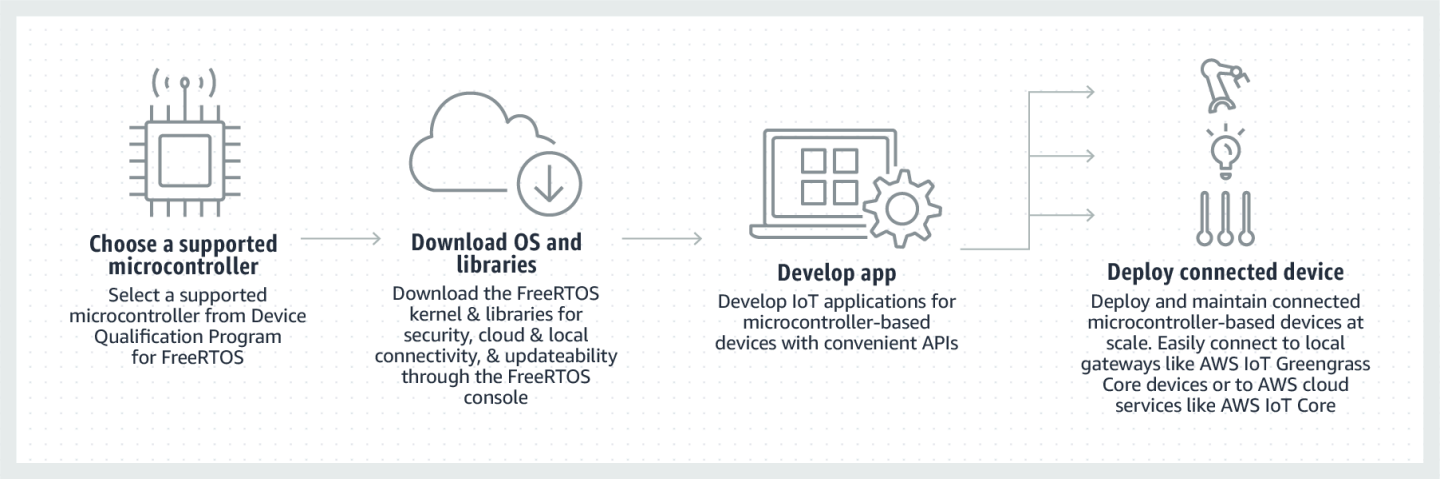
- Kit Cypress CY8CKIT-064S0S2-4343W
- Espressif ESP32- C DevKit
- Espressif ESP-WROVER-KIT
- Espressif ESP-WROOM-32SE
- Espressif ESP32-S2-Saola-1
- Kit de conectividad de IoT de Infineon XMC4800
- Kit de inicio de AWS IoT de Marvell MW320
- Kit de inicio de AWS IoT de Marvell MW322
- MediaTek Kit de desarrollo MT7697hx
- Paquete Microchip Curiosity PIC32MZEF
- Nordic nRF52840-DK
- NuMaker-IoT-M487
- Módulo de IoT LPC54018 de NXP
- Solución de seguridad OPTIGA Trust X
- Módulo de IoT Renesas RX65N RSK
- Nodo de IoT de kit de detección STMicroelectronicsSTM32L4
- Texas Instruments CC3220SF-LAUNCHXL
- Microsoft Windows 7 o posterior, con al menos una conexión de núcleo doble y una conexión de Ethernet alámbrica
- Kit IoT industrial Xilinx Avnet MicroZed

Los dispositivos cualificados también se incluyen en el AWS Partner Device Catalog.

Para obtener información acerca de la calificación de un dispositivo nuevo, consulte la Guía de calificación de FreeRTOS.

Flujo de trabajo de desarrollo

Puede iniciar el desarrollo descargando FreeRTOS. Tiene que descomprimir el paquete e importarlo a su IDE. A continuación, puede desarrollar una aplicación integrada en su plataforma de hardware seleccionada, y fabricar e implementar estos dispositivos mediante el proceso de desarrollo adecuado para su dispositivo. Los dispositivos implementados pueden conectarse al servicio de AWS IoT o AWS IoT Greengrass como parte de una solución de IoT completa.



Recursos adicionales

Estos recursos pueden resultarle útiles.

- Para obtener documentación sobre FreeRTOS adicional, consulte freertos.org.
- Si tiene preguntas sobre FreeRTOS para el equipo de ingeniería de FreeRTOS, puede abrir una edición en la página de FreeRTOS. GitHub
- Para preguntas técnicas sobre FreeRTOS, visite los Foros de la comunidad de FreeRTOS.
- Para obtener más información sobre cómo conectar dispositivos a AWS IoT, consulte Aprovisionamiento de dispositivos en la Guía para desarrolladores de AWS IoT Core.
- Para obtener asistencia técnica para AWS, consulte el Centro de soporte de AWS.
- Si tiene alguna pregunta acerca de la facturación de AWS, los servicios de cuentas, eventos, abuso u otros problemas con AWS, visite la página Contacte con nosotros.

Aspectos fundamentales del kernel de FreeRTOS

El kernel de FreeRTOS es un sistema operativo en tiempo real que admite numerosas arquitecturas. Es ideal para crear aplicaciones de microcontroladores integradas. Proporciona:

- Un programador multitareas.
- Varias opciones de asignación de memoria (incluida la opción de crear sistemas asignados de forma totalmente estática).
- Primitivos de coordinación entre tareas, como notificaciones de tareas, colas de mensajes, varios tipos de semáforo y búferes de transmisión y mensajes.
- Compatibilidad para el multiprocesamiento simétrico (SMP) en microcontroladores de varios núcleos.

El kernel de FreeRTOS nunca realiza operaciones no deterministas, como, por ejemplo, recorrer una lista enlazada, dentro de interrupciones o secciones críticas. El kernel de FreeRTOS incluye una implementación de temporizador de software eficiente que no utiliza tiempo de CPU a menos que el temporizador necesite mantenimiento. Las tareas bloqueadas no necesitan laborioso mantenimiento periódico. Las notificaciones directas a la tarea facilitan una rápida señalización de la tarea, sin casi ningún gasto de RAM. Se pueden utilizar en la mayoría de las situaciones de señalización entre tareas y de interrupción a tarea.

El kernel de FreeRTOS cuenta con un diseño pequeño, sencillo y fácil de usar. Una imagen binaria típica del kernel de RTOS se encuentra en el rango de 4000 a 9000 bytes.

Para obtener la documentación más actualizada sobre el kernel de FreeRTOS, consulte FreeRTOS.org. FreeRTOS.org ofrece una serie de tutoriales detallados y guías sobre el uso del kernel FreeRTOS, incluida una Guía de inicio rápido y el documento más detallado Mastering the FreeRTOS Real Time Kernel.

Programador del kernel de FreeRTOS

Una aplicación integrada que utiliza un RTOS pueden estructurarse como un conjunto de tareas independientes. Cada tarea se ejecuta dentro de su propio contexto, sin dependencia de otras tareas. En un momento dado, solo se ejecuta una tarea en la aplicación. El programador de RTOS en tiempo real determina cuándo debe ejecutarse cada tarea. Cada tarea se proporciona con su propia pila. Cuando una tarea se intercambia para poder ejecutar otra tarea, el contexto de ejecución

de la tarea se guarda en la pila de la tarea de modo que puede restaurarse cuando esa misma tarea vuelva reanudar su ejecución más adelante.

Para proporcionar comportamiento determinista en tiempo real, el programador de tareas de FreeRTOS permite asignar prioridades estrictas a las tareas. RTOS garantiza que la tarea de máxima prioridad que pueda ejecutar reciba tiempo de procesamiento. Esto requiere compartir el tiempo de procesamiento entre tareas de la misma prioridad si están listas para ejecutarse en el mismo momento. FreeRTOS también crea una tarea de inactividad que ejecuta solo cuando no hay otras tareas listas para ejecutarse.

Administración de la memoria

En esta sección se proporciona información sobre la administración de memoria de la aplicación y la asignación de memoria del kernel.

Asignación de memoria del kernel

El kernel de RTOS necesita RAM cada vez que se crea una tarea, cola u otros objetos RTOS. La RAM se puede asignar:

- Estáticamente durante la compilación.
- Dinámicamente desde el montón de RTOS mediante funciones de creación de objetos de la API de RTOS.

Cuando se crean objetos de RTOS de forma dinámica, no siempre es adecuado usar las funciones `malloc()` y `free()` de la biblioteca C estándar por una serie de razones:

- Puede que no estén disponibles en sistemas integrados.
- Pueden ocupar valioso espacio de código.
- No son normalmente seguras para subprocesos.
- No son deterministas.

Por estas razones, FreeRTOS mantiene la API de asignación de memoria en su capa portátil. La capa portátil se encuentra fuera de los archivos de origen que implementan la funcionalidad RTOS central, de forma que podrá proporcionar una implementación específica de la aplicación adecuada para el sistema en tiempo real que está desarrollando. Cuando el kernel de RTOS necesita RAM,

llama a `pvPortMalloc()` en lugar de a `malloc()`(). Cuando se libera RAM, el kernel de RTOS llama a `vPortFree()` en lugar de a `free()`.

Administración de memoria de las aplicaciones

Cuando las aplicaciones necesitan memoria, es posible asignarla desde el montón de FreeRTOS. FreeRTOS ofrece varios esquemas de administración de montón de distinta complejidad y características. También puede proporcionar su propia implementación de montón.

El kernel de FreeRTOS incluye cinco implementaciones de montón:

`heap_1`

Es la implementación más sencilla. No permite liberar memoria.

`heap_2`

Permite liberar memoria, pero no fusiona bloques libres adyacentes.

`heap_3`

Encapsula `malloc()` y `free()` estándar para la seguridad para subprocesos.

`heap_4`

Fusiona bloques libres adyacentes para evitar la fragmentación. Incluye una opción de ubicación de dirección absoluta.

`heap_5`

Es similar a heap_4. Puede distribuir el montón por numerosas áreas de memoria no adyacentes.

Coordinación entre tareas

Esta sección contiene información sobre los primitivos de FreeRTOS.

Queues

Las colas son la principal forma de comunicación entre tareas. Pueden utilizarse para enviar mensajes entre tareas y entre interrupciones y tareas. En la mayoría de los casos, se utilizan como búferes FIFO (primero en entrar, primero en salir) seguros para subprocesos y los datos nuevos se envían al final de la cola. (Los datos también se puede enviar al principio de la cola). Los mensajes

se envían a través de colas mediante copia, lo que significa que los datos (que pueden ser un puntero a búferes de mayor tamaño) en sí se copian en la cola, en lugar de limitarse a almacenar una referencia a los datos.

Las API de cola permiten especificar un tiempo de bloqueo. Cuando una tarea intenta leer una cola vacía, la tarea se coloca en estado bloqueado hasta que haya datos disponibles en la cola o hasta que transcurra el tiempo de bloqueo. Las tareas en estado bloqueado no consumen tiempo de CPU, lo que permite ejecutar otras tareas. Asimismo, cuando una tarea intenta escribir en una cola vacía, la tarea se coloca en estado bloqueado hasta que haya espacio disponible en la cola o hasta que transcurra el tiempo de bloqueo. Si hay más de una tarea bloqueada en la misma cola, se desbloquea primero la tarea con la prioridad más alta.

Otros primitivos de FreeRTOS, como, por ejemplo, las notificaciones directas a la tarea y los búferes de mensajes y transmisión, ofrecen alternativas ligeras a las colas en muchas situaciones de diseño habituales.

Semáforos y exclusiones mutuas

El kernel de FreeRTOS proporciona semáforos binarios, semáforos de recuento y exclusiones mutuas con fines de exclusión mutua y sincronización.

Los semáforos binarios solo pueden tener dos valores. Son una buena opción para implementaciones de sincronización (ya sea entre tareas o entre tareas e interrupciones). Los semáforos de recuento tienen más de dos valores. Permiten compartir recursos entre muchas tareas o realizar las operaciones de sincronización más complejas.

Las exclusiones mutuas son semáforos binarios que incluyen un mecanismo de herencia de prioridades. Esto significa que si una tarea de alta prioridad se bloquea al intentar obtener una exclusión mutua que actualmente está en manos de una tarea de menor prioridad, la prioridad de la tarea que tiene el token se eleva temporalmente a la de la tarea bloqueada. Este mecanismo está diseñado para garantizar que la tarea de mayor prioridad se mantenga en estado bloqueado el menor tiempo posible, a fin de minimizar la inversión de prioridades que ha tenido lugar.

Notificaciones directas a la tarea

Las notificaciones de tareas permiten a las tareas interactuar con otras tareas así como sincronizarse con rutinas de servicio de interrupción (ISR), sin necesidad de un objeto de comunicación independiente como un semáforo. Cada tarea de RTOS tiene un valor de notificación de 32 bits que se utiliza para almacenar el contenido de la notificación, de haberlo. Una notificación de tarea de

RTOS es un evento enviado directamente a una tarea que puede desbloquear la tarea receptora y, de forma opcional, actualizar el valor de notificación de la tarea receptora.

Las notificaciones de tareas de RTOS se pueden utilizar como alternativa ligera y rápida a los semáforos de recuento y binarios y, en algunos casos, a las colas. Las notificaciones de tareas ofrecen ventajas de velocidad y huella de RAM con respecto a las otras características de FreeRTOS que se pueden utilizar para realizar una funcionalidad equivalente. Sin embargo, las notificaciones de tareas únicamente se pueden utilizar cuando solo hay una tarea que puede ser receptora del evento.

Búferes de transmisión

Los búferes de transmisión permiten pasar una secuencia de bytes de una rutina de servicio de interrupción a una tarea o de una tarea a otra. Una secuencia de bytes puede tener una longitud arbitraria y no tiene que tener un principio o un final. Es posible escribir cualquier número de bytes en un momento dado y es posible leer cualquier número de bytes en un momento dado. La funcionalidad de búfer de transmisión se habilita al incluir el archivo de origen `stream_buffer.c` en su proyecto.

Los búferes de transmisión dan por hecho que solo hay una tarea o interrupción que escribe en el búfer (el escritor) y solo una tarea o interrupción que lee del búfer (el lector). Es seguro que el escritor y el lector sean distintas tareas o rutinas de servicio de interrupción, pero no es seguro tener varios escritores o lectores.

La implementación del búfer de transmisión utiliza notificaciones directas a las tareas. Por lo tanto, llamar a una API de búfer de transmisión que coloca la tarea que llama en estado bloqueado puede cambiar el valor y el estado de la notificación de la tarea que llama.

Envío de datos

`xStreamBufferSend()` se utiliza para enviar datos a un búfer de transmisión en una tarea. `xStreamBufferSendFromISR()` se utiliza para enviar datos a un búfer de transmisión en una rutina de servicio de interrupción (ISR).

`xStreamBufferSend()` permite especificar un tiempo de bloqueo. Si `xStreamBufferSend()` se llama con un tiempo de bloqueo distinto de cero para escribir en un búfer de transmisión y el búfer está lleno, la tarea se coloca en estado bloqueado hasta que haya espacio disponible o transcurra el tiempo de bloqueo.

`sbSEND_COMPLETED()` y `sbSEND_COMPLETED_FROM_ISR()` son macros que la API de FreeRTOS llama internamente cuando se escriben datos en un búfer de transmisión. Se necesita el controlador

del búfer de transmisión que se ha actualizado. Ambas macros comprueban si hay una tarea bloqueada en el búfer de transmisión a la espera de datos y, de ser así, eliminan la tarea del estado bloqueado.

Para cambiar este comportamiento predeterminado, proporcione su propia implementación de `sbSEND_COMPLETED()` en `FreeRTOSConfig.h`. Esto resulta útil cuando se utiliza un búfer de transmisión para transferir datos entre núcleos en un procesador de varios núcleos. En ese caso, se puede implementar `sbSEND_COMPLETED()` para generar una interrupción en el otro núcleo de la CPU y la rutina del servicio de interrupción puede entonces utilizar la API `xStreamBufferSendCompletedFromISR()` para comprobar y, si es necesario, desbloquear, una tarea que está a la espera de datos.

Recepción de datos

`xStreamBufferReceive()` se utiliza para leer datos de un búfer de transmisión en una tarea. `xStreamBufferReceiveFromISR()` se utiliza para leer datos de un búfer de transmisión en una rutina de servicio de interrupción (ISR).

`xStreamBufferReceive()` permite especificar un tiempo de bloqueo. Si `xStreamBufferReceive()` se llama con un tiempo de bloqueo distinto de cero para leer de un búfer de transmisión y el búfer está vacío, la tarea se coloca en estado bloqueado hasta que esté disponible una determinada cantidad de datos en el búfer de transmisión o hasta que transcurra el tiempo de bloqueo.

La cantidad de datos que debe haber en el búfer de transmisión antes de que se desbloquee la tarea se denomina umbral de activación del búfer de transmisión. Una tarea bloqueada con un nivel de activación de 10 se desbloquea cuando se escriben al menos 10 bytes en el búfer o cuando transcurre el tiempo de bloqueo. Si el tiempo de bloqueo de una tarea de lectura caduca antes de que se alcance el nivel de activación, la tarea recibe cualquier dato escrito en el búfer. El nivel de activación de la tarea se debe establecer en un valor entre 1 y el tamaño del búfer de transmisión. El nivel de activación del búfer de transmisión se establece cuando se llama a `xStreamBufferCreate()`. Para cambiarlo, llame a `xStreamBufferSetTriggerLevel()`.

`sbRECEIVE_COMPLETED()` y `sbRECEIVE_COMPLETED_FROM_ISR()` son macros que la API de FreeRTOS llama internamente cuando se leen datos de un búfer de transmisión. Los macros comprueban si hay una tarea bloqueada en el búfer de transmisión que está esperando a que haya espacio para volverse disponibles dentro del búfer y, de ser así, eliminan la tarea del estado bloqueado. Para cambiar el comportamiento predeterminado de `sbRECEIVE_COMPLETED()` al proporcionar una implementación alternativa en `FreeRTOSConfig.h`.

Búferes de mensajes

Los búferes de mensajes permiten pasar mensajes discretos de longitud variable de una rutina de servicio de interrupción a una tarea o de una tarea a otra. Por ejemplo, los mensajes de 10, 20 y 123 bytes de longitud se pueden escribir o leer en el mismo búfer de mensajes. Los mensajes de 10 bytes solo se puede leer como mensajes de 10 bytes, no como bytes individuales. Los búferes de mensajes se basan en la implementación del búfer de transmisión. Puede habilitar la funcionalidad del búfer de mensajes al incluir el archivo de origen `stream_buffer.c` en su proyecto.

Los búferes de mensajes dan por hecho que solo hay una tarea o interrupción que escribe en el búfer (el escritor) y solo una tarea o interrupción que lee del búfer (el lector). Es seguro que el escritor y el lector sean distintas tareas o rutinas de servicio de interrupción, pero no es seguro tener varios escritores o lectores.

La implementación del búfer de mensajes utiliza notificaciones directas a las tareas. Por lo tanto, llamar a una API de búfer de transmisión que coloca la tarea que llama en estado bloqueado puede cambiar el valor y el estado de la notificación de la tarea que llama.

Para permitir que los búferes de mensajes controlen mensajes de tamaños variables, la longitud de cada mensaje se escribe en el búfer de mensajes antes que el mensaje en sí. La longitud se almacena en una variable de tipo `size_t`, que suelen ser 4 bytes en una arquitectura de 32 bytes. Por lo tanto, al escribir un mensaje de 10 bytes en el búfer de mensajes, se consumen realmente 14 bytes de espacio del búfer. Asimismo, al escribir un mensaje de 100 bytes en el búfer de mensajes, se usan realmente 104 bytes de espacio del búfer.

Envío de datos

`xMessageBufferSend()` se utiliza para enviar datos a un búfer de mensajes desde una tarea. `xMessageBufferSendFromISR()` se utiliza para enviar datos a un búfer de mensajes desde una rutina de servicio de interrupción (ISR).

`xMessageBufferSend()` permite especificar un tiempo de bloqueo. Si `xMessageBufferSend()` se llama con un tiempo de bloqueo distinto de cero para escribir en un búfer de mensajes y el búfer está lleno, la tarea se coloca en estado bloqueado hasta que haya espacio disponible en el búfer de mensajes o transcurra el tiempo de bloqueo.

`sbSEND_COMPLETED()` y `sbSEND_COMPLETED_FROM_ISR()` son macros que la API de FreeRTOS llama internamente cuando se escriben datos en un búfer de transmisión. Se necesita un único parámetro, que es el controlador del búfer de transmisión que se ha actualizado. Ambas macros

comprueban si hay una tarea bloqueada en el búfer de transmisión a la espera de datos y, de ser así, eliminan la tarea del estado bloqueado.

Para cambiar este comportamiento predeterminado, proporcione su propia implementación de `sbSEND_COMPLETED()` en `FreeRTOSConfig.h`. Esto resulta útil cuando se utiliza un búfer de transmisión para transferir datos entre núcleos en un procesador de varios núcleos. En ese caso, se puede implementar `sbSEND_COMPLETED()` para generar una interrupción en el otro núcleo de la CPU y la rutina del servicio de interrupción puede entonces utilizar la API `xStreamBufferSendCompletedFromISR()` para comprobar y, si es necesario, desbloquear, una tarea que estaba a la espera de datos.

Recepción de datos

`xMessageBufferReceive()` se utiliza para leer datos de búfer de mensajes en una tarea. `xMessageBufferReceiveFromISR()` se utiliza para leer datos de un búfer de mensajes en una rutina de servicio de interrupción (ISR). `xMessageBufferReceive()` permite especificar un tiempo de bloqueo. Si `xMessageBufferReceive()` se llama con un tiempo de bloqueo distinto de cero para leer de un búfer de mensajes y el búfer está vacío, la tarea se coloca en estado bloqueado hasta que haya datos disponibles o transcurra el tiempo de bloqueo.

`sbRECEIVE_COMPLETED()` y `sbRECEIVE_COMPLETED_FROM_ISR()` son macros que la API de FreeRTOS llama internamente cuando se leen datos de un búfer de transmisión. Los macros comprueban si hay una tarea bloqueada en el búfer de transmisión que está esperando a que haya espacio para volverse disponibles dentro del búfer y, de ser así, eliminan la tarea del estado bloqueado. Para cambiar el comportamiento predeterminado de `sbRECEIVE_COMPLETED()` al proporcionar una implementación alternativa en `FreeRTOSConfig.h`.

Compatibilidad para el multiprocesamiento simétrico (SMP)

La compatibilidad para SMP en el kernel de FreeRTOS permite que una instancia del kernel de FreeRTOS programe tareas en varios núcleos de procesador idénticos. Las arquitecturas del núcleo deben ser idénticas y compartir la misma memoria.

Modificación de aplicaciones para usar el kernel de FreeRTOS-SMP

La API de FreeRTOS sigue siendo prácticamente la misma entre las versiones de un solo núcleo y SMP, a excepción de estas API adicionales. Por lo tanto, una aplicación escrita para la versión de un solo núcleo de FreeRTOS debería compilarse con la versión de SMP con un esfuerzo mínimo o nulo.

Sin embargo, es posible que haya algunos problemas funcionales, ya que algunas suposiciones que eran ciertas para las aplicaciones de un solo núcleo pueden dejar de serlo para las aplicaciones de varios núcleos.

Una suposición común es que una tarea de menor prioridad no se puede ejecutar mientras se está ejecutando una tarea de mayor prioridad. Si bien esto era cierto en un sistema de un solo núcleo, ya no lo es en los sistemas de varios núcleos, ya que se pueden ejecutar varias tareas simultáneamente. Si la aplicación se basa en las prioridades relativas de las tareas para excluirse mutuamente, podría observar resultados inesperados en un entorno multinúcleo.

Otra suposición común es que los ISR no pueden ejecutarse simultáneamente entre sí ni con otras tareas. Esto ya no es cierto en un entorno multinúcleo. El redactor de la aplicación debe garantizar una exclusión mutua adecuada al acceder a los datos compartidos entre las tareas y los ISR.

Temporizadores de software

Un temporizador de software permite ejecutar una función en un momento determinado en el futuro. La función ejecutada por el temporizador se denomina función de devolución de llamada del temporizador. El tiempo entre el inicio de un temporizador y la ejecución de la función de devolución de llamada se denomina periodo del temporizador. El kernel de FreeRTOS proporciona una implementación de temporizador de software eficiente porque:

- No ejecuta funciones de devolución de llamada del temporizador desde un contexto de interrupción.
- No consume tiempo de procesamiento a menos que el temporizador haya caducado.
- No añade gastos de procesamiento a la interrupción de ciclo.
- No recorre estructuras de listas de enlace si las interrupciones están deshabilitadas.

Soporte de bajo consumo

Al igual que la mayoría de los sistemas operativos integrados, el kernel de FreeRTOS utiliza un temporizador de hardware para generar interrupciones de ciclo periódicas, que se utilizan para medir el tiempo. El ahorro energético de las implementaciones de temporizador de hardware normales está limitado por la necesidad de salir y volver a entrar, de forma periódica, en el estado de bajo consumo para procesar interrupciones de ciclo. Si la frecuencia de la interrupción de ciclo es demasiado alta, la energía y el tiempo consumidos en entrar y salir del estado de bajo consumo para cada ciclo

superan los posibles ahorros energéticos obtenidos en todos los casos salvo en los modos de ahorro de energía más ligeros.

Para solucionar esta limitación, FreeRTOS incluye un modo de temporizador sin ciclo para las aplicaciones de bajo consumo. El modo inactivo sin ciclo de FreeRTOS detiene la interrupción de ciclo periódica durante los periodos de inactividad (cuando no hay tareas de la aplicación que se puedan ejecutar) y, a continuación, realiza un ajuste de corrección del valor de recuento de ciclo de RTOS cuando se reinicia la interrupción de ciclo. La detención de la interrupción de ciclo permite que el microcontrolador permanezca en un estado de ahorro de energía profundo hasta que se produzca una interrupción o llegue el momento de que el kernel de RTOS pase una tarea al estado listo.

Configuración del kernel

Puede configurar el kernel de FreeRTOS para una placa y una aplicación específicas con el archivo de encabezado `FreeRTOSConfig.h`. Cada aplicación creada en el kernel debe tener un archivo de encabezado `FreeRTOSConfig.h` en la ruta de inclusión del preprocesador. `FreeRTOSConfig.h` es específico de la aplicación y se debería colocar en un directorio de aplicaciones y no en uno de directorios de códigos fuente del kernel de FreeRTOS.

Los archivos `FreeRTOSConfig.h` de las aplicaciones de demostración y prueba de se encuentran en *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/FreeRTOSConfig.h` y *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_tests/config_files/FreeRTOSConfig.h`.

Para obtener una lista de los parámetros de configuración disponibles para especificarlos en `FreeRTOSConfig.h`, consulte FreeRTOS.org

SDK de dispositivos de AWS IoT para Embedded C

> **Note**
>
> Este SDK está diseñado para que lo utilicen desarrolladores de software incrustado con experiencia.

El AWS IoT Device SDK para Embedded C (C-SDK) es un conjunto de archivos de origen C con licencia de código abierto del MIT, que se puede utilizar en aplicaciones integradas para establecer conexiones seguras con dispositivos IoT en AWS IoT Core. Incluye un cliente MQTT, un cliente HTTP, un analizador JSON y bibliotecas de sombra de dispositivo de AWS IoT, de trabajos de AWS IoT, de aprovisionamiento de flota de AWS IoT y de AWS IoT Device Defender. Este SDK se distribuye como código fuente y puede integrarse en el firmware del cliente junto con código de aplicación, otras bibliotecas y un sistema operativo (OS) de su elección.

AWS IoT Device SDK para Embedded C generalmente se dirige a dispositivos con limitaciones de recursos que requieren un tiempo de ejecución optimizado del lenguaje C. Puede usar el SDK en cualquier sistema operativo y alojarlo en cualquier tipo de procesador (por ejemplo, MCU y MPU). Sin embargo, si sus dispositivos tienen suficientes recursos de memoria y procesamiento, le recomendamos que utilice uno de los SDK de dispositivo de AWS IoT de orden superior.

Para obtener más información, consulte lo siguiente:

- SDK de dispositivos de AWS IoT para C integrado
- AWS IoT Device SDK para Embedded C en GitHub
- AWS IoT Device SDK for Embedded C Readme
- Ejemplos de SDK de dispositivos de AWS IoT para C integrado

E/S común

Las API de E/S comunes actúan como capas de abstracción de hardware (HAL) y proporcionan una interfaz común entre controladores y código de aplicación de nivel superior. La E/S común de FreeRTOS proporciona un conjunto de API estándar para acceder a dispositivos serie comunes en placas de referencia compatibles; las implementaciones de estas API no están incluidas. Estas API comunes se comunican e interactúan con estos periféricos y permiten que el código funcione en todas las plataformas. Sin E/S común, el código para dispositivos de bajo nivel es específico del proveedor de hardware.

> **Note**
>
> FreeRTOS no requiere la implementación de las API de E/S comunes para funcionar, pero intentará utilizar las API de E/S comunes como una forma de interactuar con los periféricos específicos de una placa basada en un microcontrolador en lugar de con las API específicas del proveedor.

En general, los controladores de dispositivo son independientes del sistema operativo subyacente y son específicos de una configuración de hardware determinada. La capa HAL abstrae los detalles de cómo funciona un controlador específico y proporciona una API uniforme para controlar dichos dispositivos. Puede utilizar las mismas API para obtener acceso a varios controladores de dispositivo a través de placas de referencia basadas en varios microcontroladores (MCU).

Bibliotecas

Actualmente, FreeRTOS proporciona dos bibliotecas de E/S comunes: E/S común (básica) y E/S común (BLE).

E/S común - Básica

Información general

E/S común - Básica proporciona API que se ocupan de los periféricos y funciones de E/S básicos que se pueden encontrar en las placas basadas en MCU. El repositorio de E/S común - Básica está disponible en GitHub.

Periféricos admitidos

- ADC
- GPIO
- I2C
- PWM
- SPI
- UART
- Watchdog
- Flash
- RTC
- EFUSE
- Restablecimientos
- I2S
- Contador de rendimiento
- Información sobre la plataforma de hardware

Características admitidas

- Lectura y escritura sincrónicas

  La función no regresa hasta que se transfiere la cantidad de datos solicitada.

- Lectura y escritura asincrónicas

  La función regresa inmediatamente y la transferencia de datos se realiza de forma asíncrona. Cuando se completa la acción, se invoca una devolución de llamada de usuario registrado.

Código específico del periférico

- I2C

  Combina varias operaciones en una sola transacción. Se utiliza para acciones de escritura y luego lectura en una sola transacción.

- SPI

  Transfiere datos entre la principal y secundaria, lo que significa que la escritura y la lectura se realizan simultáneamente.

Referencia de la API

Para obtener una referencia completa sobre la API, consulte la referencia sobre la API E/S común - Básica.

E/S común - BLE

Información general

E/S común - BLE proporciona una abstracción de la pila de Bluetooth de bajo consumo del fabricante. Proporciona las siguientes interfaces que se pueden utilizar para controlar el dispositivo y realizar operaciones GAP y GATT. El repositorio de E/S común - BLE está disponible en GitHub.

Administrador de dispositivos Bluetooth:

Proporciona una interfaz para controlar el dispositivo Bluetooth, realizar operaciones de detección de dispositivos y otras tareas relacionadas con la conectividad.

Administrador de adaptadores BLE:

Proporciona una interfaz para las funciones de la API de GAP que son específicas de BLE.

Administrador de adaptadores Bluetooth clásico:

Proporciona una interfaz para controlar las funcionalidades de BT clásico de un dispositivo.

Servidor de GATT:

Proporciona una interfaz para utilizar la característica de servidor GATT de Bluetooth.

Cliente de GATT:

Proporciona una interfaz para utilizar la característica de cliente de GATT de Bluetooth.

Interfaz de conexión A2DP:

Proporciona una interfaz para el perfil de fuente A2DP del dispositivo local.

Referencia de la API

Para obtener una referencia completa sobre la API, consulte la referencia sobre la API E/S común - BLE.

E/S común para software común de Amazon

Las API de E/S común forman parte de las implementaciones requeridas por Amazon Common Software para dispositivos, específicamente para implementarlas en un kit de portabilidad de dispositivos (DPK) de un proveedor.

¿Qué es ACS?

Amazon Common Software (ACS) para dispositivos es un software que agiliza la integración de los SDK de dispositivos Amazon en sus dispositivos. ACS proporciona una capa de integración de API unificada, componentes previamente validados y con uso eficiente de la memoria para funciones comunes como la conectividad, un kit de portabilidad de dispositivos (DPK) y conjuntos de pruebas de varios niveles.

Programa de calificación

El programa de calificación Amazon Common Software para dispositivos verifica que una versión del DPK (kit de portabilidad de dispositivos) de ACS que se ejecuta en una placa de desarrollo específica basada en un microcontrolador sea compatible con las prácticas recomendadas publicadas por el programa y lo suficientemente sólida como para superar las pruebas exigidas por ACS especificadas en el programa de calificación.

Los proveedores que cumplen los requisitos de este programa figuran en la página de proveedores de chipsets de ACS.

Para obtener más información sobre la calificación, consulte ACS para dispositivos.

Introducción a FreeRTOS

Temas:

Introducción a AWS IoT y FreeRTOS con Quick Connect

Para explorar rápidamente AWS IoT, comience con las Demostraciones de AWS Quick Connect. Las demostraciones de Quick Connect son fáciles de configurar y conectan una placa calificada FreeRTOS proporcionada por un socio a AWS IoT.

Siga el tutorial Introducción a AWS IoT para comprender mejor AWS IoT y la consola de AWS IoT. Puede modificar el código fuente de la demostración que se proporciona con las demostraciones de Quick Connect utilizando el sistema de creación y las herramientas de la placa elegidos para conectarse a su cuenta de AWS. El flujo de datos de la consola de AWS IoT de su cuenta ya está visible.

Exploración de las bibliotecas de FreeRTOS

Una vez que comprenda cómo funcionan juntos un dispositivo de IoT e AWS IoT, puede empezar a explorar las bibliotecas de FreeRTOS y las bibliotecas de soporte a largo plazo (LTS).

Algunas de las bibliotecas más utilizadas para los dispositivos AWS IoT basados en FreeRTOS son:

- Kernel de FreeRTOS
- coreMQTT
- Inalámbricas (OTA) de AWS IoT

Visite freertos.org para ver documentación técnica y demostraciones específicas de la biblioteca.

Obtenga información sobre cómo crear un producto de AWS IoT seguro y sólido

Consulte las integraciones de AWS IoT destacadas de FreeRTOS para obtener información sobre las prácticas recomendadas para hacer que el software de los dispositivos IoT sea más seguro y sólido. Estas integraciones de IoT de FreeRTOS están diseñadas para mejorar la seguridad mediante una combinación del software FreeRTOS y una placa proporcionada por un socio con funciones de seguridad de hardware. Puede utilizarlas en producción tal cual o como modelo para sus propios diseños.

Desarrollo de su producto de aplicación de AWS IoT

Siga estos pasos para crear un proyecto de aplicación para su producto de AWS IoT:

1. Descargue la última versión de FreeRTOS o soporte a largo plazo (LTS) desde freertos.org o clónela desde el repositorio de GitHub FreeRTOS-LTS. También puede integrar las bibliotecas FreeRTOS necesarias en su proyecto desde la cadena de herramientas del proveedor de MCU, si están disponibles.
2. Siga la Guía de portabilidad de FreeRTOS para crear un proyecto, configurar el entorno de desarrollo e integrar las bibliotecas de FreeRTOS en su proyecto. Use el repositorio de GitHub FreeRTOS-Libraries-Integration-Tests para validar la portabilidad.

AWS IoT Device Tester para FreeRTOS

El IDT para FreeRTOS es una herramienta para calificar la tasa de rendimiento de datos con el sistema operativo FreeRTOS. El comprobador de dispositivos (IDT) abre primero una conexión USB o UART a un dispositivo. A continuación, muestra una imagen de FreeRTOS configurada para probar la funcionalidad del dispositivo en diversas condiciones. Los conjuntos de AWS IoT Device Tester son ampliables y el IDT se utiliza para la orquestación de las pruebas de AWS IoT del cliente.

IDT para FreeRTOS se ejecuta en un equipo host (Windows, macOS o Linux) que esté conectado al dispositivo que se tiene que probar. IDT configura y orquesta los casos de prueba y agrega los resultados. También proporciona una interfaz de línea de comandos para administrar la ejecución de las pruebas.

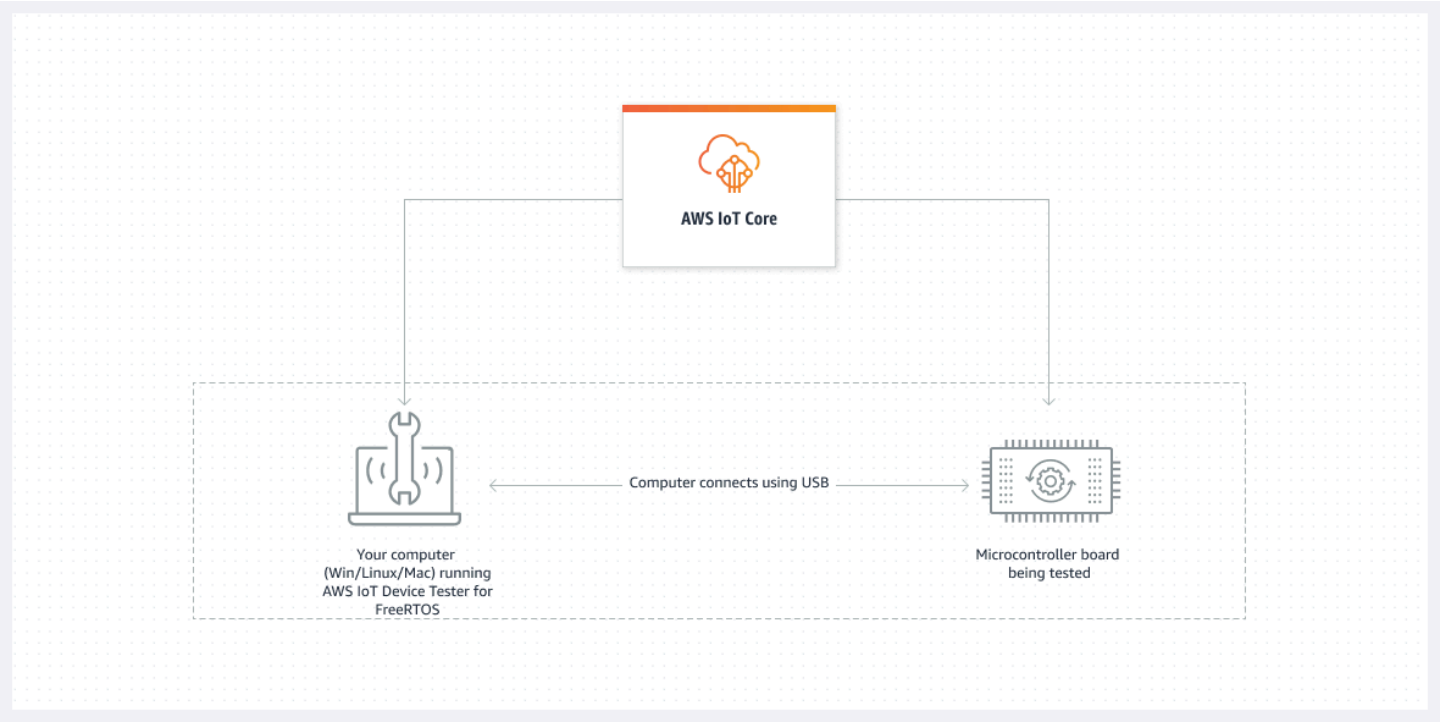
Conjunto de calificación de FreeRTOS

IDT para FreeRTOS verifica la portabilidad de FreeRTOS en su microcontrolador y comprueba si se puede comunicar eficazmente con AWS IoT de forma fiable y segura. En concreto, verifica que las interfaces de la capa de portabilidad de las bibliotecas de FreeRTOS se implementan correctamente. También realiza pruebas integrales con AWS IoT Core. Por ejemplo, verifica si la placa es capaz de enviar y recibir mensajes MQTT y procesarlos correctamente.

El conjunto de calificación de FreeRTOS (FRQ) 2.0 utiliza casos de prueba de FreeRTOS-Libraries-Integration-Tests y Device Advisor definidos en la Guía de calificación de FreeRTOS.

IDT para FreeRTOS genera informes de pruebas que puede enviar a la Red de socios de AWS (APN) para incluir sus dispositivos FreeRTOS en el Catálogo de dispositivos de socios de AWS. Para obtener más información, consulte el Programa de cualificación de dispositivos de AWS.

En el siguiente diagrama se muestra la configuración de la infraestructura de las pruebas para la calificación de FreeRTOS.



IDT para FreeRTOS organiza los recursos de 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 FreeRTOS.
- Un grupo de pruebas es el conjunto de pruebas individuales relacionadas con una característica particular, como la mensajería BLE y MQTT.

Para obtener más información, consulte Versiones del conjunto de pruebas

Compatibilidad con los conjuntos de prueba

IDT para FreeRTOS combina una configuración de ajustes estándar y un formato de resultados con un entorno de pruebas. Este entorno 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.

La forma en que configure los conjuntos de pruebas personalizados determina las configuraciones de los ajustes que debe proporcionar a sus usuarios para ejecutar los conjuntos de pruebas personalizados. Para obtener más información, consulte Uso de IDT para desarrollar y ejecutar sus propios conjuntos de pruebas.

Versiones compatibles de AWS IoT Device Tester para FreeRTOS

En este tema se muestran las versiones compatibles de AWS IoT Device Tester para FreeRTOS. Como práctica recomendada, le recomendamos que utilice la versión más reciente de IDT para FreeRTOS que sea compatible con la versión de destino de FreeRTOS. Cada versión de IDT para FreeRTOS tiene una o varias versiones correspondientes de FreeRTOS compatibles. Le recomendamos que descargue una nueva versión de IDT para FreeRTOS cuando se publique una nueva versión de FreeRTOS.

Al descargar el software, acepta el contrato de licencia de AWS IoT Device Tester incluido en el archivo de descarga.

> **Note**
>
> Cuando utilice AWS IoT Device Tester para FreeRTOS, le recomendamos que actualice al último parche de la versión más reciente de FreeRTOS-LTS.

> **Important**
>
> Desde octubre de 2022, AWS IoT Device Tester para la calificación de AWS IoT para FreeRTOS (FRQ) 1.0 no genera informes de calificación firmados. No puede calificar nuevos dispositivos de FreeRTOS de AWS IoT para incluirlos en el Catálogo de dispositivos de socios de AWS a través del Programa de calificación de dispositivos de AWS con las versiones IDT FRQ 1.0. Si bien no puede calificar los dispositivos FreeRTOS con IDT FRQ 1.0, puede seguir probando los dispositivos FreeRTOS con FRQ 1.0. Le recomendamos que utilice IDT FRQ 2.0 para calificar y enumerar los dispositivos FreeRTOS en el Catálogo de dispositivos de socios de AWS.

Última versión de AWS IoT Device Tester para FreeRTOS

Utilice los siguientes enlaces para descargar las versiones más recientes de IDT para FreeRTOS.

Última versión de AWS IoT Device Tester para FreeRTOS

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
IDT v4.9.0	FRQ_2.5.0	• 202112.00 • 202212.00 • 202212.01 • Todos los parches de FreeRTOS 202210-LTS que utilizan las bibliotecas de FreeRTOS LTS.	• Linux • macOS • Windows	04/04/2023	• Admite las pruebas con FreeRTOS 202112, 202212, 202212.01 y todos los parches de FreeRTOS 202210-LTS que utilizan bibliotecas FreeRTOS. Para obtener más información, consulte el archivo README.md. Debe incluir la versión de parche para FreeRTOS-LTS en su

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
					`manifest.yml` . • Tiempo de ejecución mejorado de las pruebas E2E OTA. • Limita la cantidad de dispositivos enumerados en `device.json` a 1. • Pequeñas correcciones de errores y mejoras.

> **Note**
>
> No se recomienda que varios usuarios ejecuten IDT desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Esto puede provocar bloqueos o daños en los datos. Le recomendamos que extraiga el paquete IDT en una unidad local y ejecute el binario IDT en su estación de trabajo local.

Versiones anteriores de IDT para FreeRTOS

También se admiten las siguientes versiones anteriores de IDT para FreeRTOS.

Versiones anteriores de AWS IoT Device Tester para FreeRTOS

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
IDT v4.8.1	FRQ_2.4.0	• 202112.00 • 202212.00 • 202212.01 • Todos los parches de FreeRTOS 202210-LTS que utilizan las bibliotecas de FreeRTOS LTS.	• Linux • macOS • Windows	23/01/2023	• Para obtener más información, consulte el archivo README.md. Debe incluir la versión de parche para FreeRTOS-LTS en su `manifest.yml` . • Pequeñas correcciones de errores y mejoras.
IDT v4.6.0	FRQ_2.3.0	• 202112.00 • 202212.00 • 202212.01 • 202210-LTS que utilizan bibliotecas	• Linux • macOS • Windows	16/11/2022	• Para obtener más información, consulte el archivo README.md

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
		FreeRTOS LTS.			. Debe incluir la versión de parche para FreeRTOS-LTS en su `manifest.yml` . • Para obtener más información acerca de lo que se incluye en la versión 202210-LTS de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub. • Añade la capacidad de configurar y ejecutar AWS IoT Device

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
					Tester para FreeRTOS a través de una interfaz de usuario basada en web. Para empezar, consulte Uso de la interfaz de usuario de IDT para FreeRTOS para ejecutar el conjunto de calificación FreeRTOS 2.0 (FRQ 2.0). • Añade una opción para conservar las copias modificadas del código fuente creado y utilizado en

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
					tiempo de ejecución para la depuración posterior a las pruebas. Para obtener más información, consulte Configurar ajustes de Build, Flash y Test. • Añade compatibilidad con el SDK del cliente de IDT para Java. Para obtener más información sobre el SDK del cliente de IDT, consulte

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
					Uso de IDT para desarrollar y ejecutar sus propios conjuntos de pruebas.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
IDT v4.5.11	FRQ_2.2.0	• 202112.00 • 202212.00 • 202212.01 • 202210-LTS que utilizan bibliotecas FreeRTOS LTS.	• Linux • macOS • Windows	14/10/2022	• Para obtener más información, consulte el archivo README.md. Debe incluir la versión de parche para FreeRTOS-LTS en su `manifest.yml` . • Para obtener más información acerca de lo que se incluye en la versión 202210-LTS de FreeRTOS, consulte el archivo CHANGELOG

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Enlaces de descarga	Fecha de publicación	Notas de la versión
					.md en GitHub. • Pequeñas correcciones de errores y mejoras.

Para obtener más información, consulte Política de compatibilidad para AWS IoT Device Tester para FreeRTOS.

Versiones de IDT no compatibles para FreeRTOS

En esta sección, se muestran las versiones de IDT para FreeRTOS que no son compatibles. Las versiones que no son compatibles no reciben actualizaciones ni correcciones de errores. Para obtener más información, consulte Política de compatibilidad para AWS IoT Device Tester para FreeRTOS.

Las siguientes versiones de IDT-FreeRTOS ya no son compatibles.

Versiones no compatibles de AWS IoT Device Tester para FreeRTOS

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v4.5.10	FRQ_2.1.4	• 202112.00 • 202012-LTS que utilizan bibliotecas FreeRTOS LTS.	02/09/2022	• Para obtener más información acerca de lo que se incluye en la versión 202012-LTS de FreeRTOS,

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				consulte el archivo CHANGELOG.md en GitHub. • Se ha resuelto un problema que afectaba al grupo de pruebas `OTA End to End`. • Se ha eliminado `FullTransportInterfacePlainText` al ejecutar las calificaciones. El texto sin formato todavía se puede ejecutar como un grupo de pruebas de desarrollo utilizando el indicador `-\-group-id` . • Se ha mejorado el registro y la

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				legibilidad de la salida de la consola y los archivos. • Pequeñas correcciones de errores y mejoras.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v4.5.9	FRQ_2.1.3	• 202112.00 • 202012.04 -LTS que utilizan bibliotecas FreeRTOS LTS.	17/08/2022	• Para obtener más información sobre lo que se incluye en la versión 202012.04 -LTS de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub. • Se ha resuelto un problema que afectaba al grupo de pruebas `FreeRTOSIntegrity` . • Se ha actualizado el grupo de pruebas `FullCloudIoT` eliminando el caso de prueba “Reintentos de retroceso exponencial

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				de conexión de MQTT”. • Pequeñas correcciones de errores y mejoras.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v4.5.6	FRQ_2.1.2	• 202112.00 • 202012.04 -LTS que utilizan bibliotecas FreeRTOS LTS.	29/06/2022	• Para obtener más información sobre lo que se incluye en la versión 202012.04 -LTS de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub. • Añade un nuevo grupo de pruebas `FullCloudIoT` con el que se evalúa la placa en AWS IoT Core Device Advisor. • Se ha resuelto un problema que afectaba a los casos de pruebas OTA E2E. • Pequeñas correcciones

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				de errores y mejoras.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v4.5.5	FRQ_2.1.1	• 202112.00 • 202012.04 -LTS que utilizan bibliotecas FreeRTOS LTS.	06/06/2022	• Para obtener más información sobre lo que se incluye en la versión 202012.04 -LTS de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub. • Añade un nuevo grupo de pruebas `FullCloudIoT` con el que se evalúa la placa en AWS IoT Core Device Advisor. • Se ha resuelto un problema que afectaba a los casos de prueba FreeRTOSVersion y FreeRTOSIntegrity.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				• Pequeñas correcciones de errores y mejoras.
IDT v4.5.5	FRQ_2.1.0	• 202107.00 • 202112.00 • 202012.04 -LTS que utilizan bibliotecas FreeRTOS LTS.	31/05/2022	• Para obtener más información sobre lo que se incluye en la versión 202012.04 -LTS de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub. • Añade un nuevo grupo de pruebas `FullCloud IoT` con el que se evalúa la placa en AWS IoT Core Device Advisor. • Pequeñas correcciones de errores y mejoras.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v4.5.4	FRQ_2.0.0	• 202112.00 • 202012.04 -LTS que utilizan bibliotecas FreeRTOS LTS.	09/05/2022	• Para obtener más información sobre lo que se incluye en la versión 202012.04 -LTS de FreeRTOS, consulte el archivo CHANGELOG .md en GitHub. • Elimina el requisito de calificar las placas utilizando únicament e versiones de Amazon FreeRTOS del repositorio de GitHub de `aws/amazo n-freerto s` . • Pequeñas correcciones de errores y mejoras.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v4.5.2	FRQ_1.6.2	202107.00	25/01/2022	• Para obtener más información sobre lo que se incluye en la versión 202107.00 de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub. • Implement a el nuevo orquestador de pruebas de IDT para configurar conjuntos de pruebas personalizados. Para obtener más información, consulte Configuración del orquestador de pruebas de IDT. • Pequeñas correcciones

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				de errores y mejoras.
IDT v4.0.3	FRQ_1.5.1	202012.00	30/07/2021	• Soporte para la calificación de dispositivos con credenciales bloqueadas en un módulo de seguridad de hardware. • Pequeñas correcciones de errores y mejoras.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v4.3.0	FRQ_1.6.1	202107.00	26/07/2021	• Para obtener más información sobre lo que se incluye en la versión 202107.00 de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub. • Añade la capacidad de configurar y ejecutar AWS IoT Device Tester para FreeRTOS a través de una interfaz de usuario basada en web. Para empezar, consulte Uso de la interfaz de usuario de IDT para FreeRTOS para ejecutar el conjunto de

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				calificación FreeRTOS.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión

IDT v4.1.0	FRQ_1.6.0	202107.00	21/07/2021	• Para obtener más información sobre lo que se incluye en la versión 202107.00 de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub.
				• Elimina los siguientes casos de prueba de la calificación OTA:
				  • Agente OTA
				  • Nombre de archivo OTA que falta
				  • Número máximo de bloques configurados de OTA
				• Elimina el grupo de pruebas `Both` del plano de

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				datos OTA de la calificación OTA. En el archivo `device.json`, la configuración de `OTADataPlaneProtocol` ahora acepta solo `HTTP` o `MQTT` como valores admitidos. • Implementa los siguientes cambios en la configuración `freertosFileConfiguration` del archivo `userdata.json` para los cambios en el código fuente de FreeRTOS: • Cambia el nombre del archivo

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				especificado para `otaAgentTestsConfig` y `otaAgentDemosConfig` de `aws_ota_agent_config.h` `aota_config.h`. • Añade una nueva configuración `otaDemosConfig` opcional para especificar la ruta del archivo `ota_demo_config.h` nuevo. • Agrega un nuevo campo `testStartDelayms` a `userdata`.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				`json` para especificar un retraso entre el momento en que se actualiza un dispositivo para ejecutar un grupo de pruebas de FreeRTOS y el momento en que comienza a ejecutar las pruebas. El valor debería estar expresado en milisegundos. Este retraso se puede utilizar para dar a IDT la oportunidad de conectarse de forma que no se pierda ninguna salida de prueba.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v4.0.1	FRQ_1.4.1	202012.00	19/01/2021	• Para obtener más información sobre lo que se incluye en la versión 202012.00 de FreeRTOS, consulte el archivo CHANGELOG.md en GitHub. • Presenta casos de prueba integrales de OTA (vía inalámbrica) adicionales. • Admite la calificación de placas de desarrollo que ejecutan FreeRTOS 202012.00 y que utilizan las bibliotecas LTS de FreeRTOS. • Añade compatibi

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				lidad con la calificación de las placas de desarroll o FreeRTOS mediante conectividad móvil. • Corrige un error en la configuración del servidor Echo. • Le permite desarrollar y ejecutar sus propios conjuntos de pruebas personali zados con AWS IoT Device Tester para FreeRTOS. Para obtener más informaci ón, consulte Uso de IDT para desarroll ar y ejecutar sus propios

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				conjuntos de pruebas. • Proporcio na aplicacio nes de IDT firmadas con código, por lo que no es necesario conceder permisos cuando se ejecuta en Windows o macOS. • Se ha perfeccionado la lógica de análisis de los resultados de las pruebas BLE.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v3.4.0	FRQ_1.3.0	202011.01	05/11/2020	• Para obtener más detalles, consulte el archivo CHANGELOG.md en GitHub. • Se ha corregido un error donde “RSA” no era una opción de configuración válida para PKCS11. • Se ha corregido un error que provocaba que los buckets de Amazon S3 no se limpiaran correctam ente tras las pruebas de OTA. • Actualiza ciones para admitir los nuevos casos de prueba dentro del

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				grupo de pruebas FullMQTT.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v3.3.0	FRQ_1.2.0	202007.00	17/09/2020	• Para obtener más detalles, consulte el archivo CHANGELOG.md en GitHub. • Nuevas pruebas integrales para validar la caracterí stica de actualización, suspensión y reanudación vía inalámbri ca (OTA). • Se ha corregido un error que provocaba que los usuarios de la región eu-central-1 no pudieran pasar la validación de configura ción para las pruebas OTA.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				• Se ha añadido el parámetro `--update-idt` al comando `run-suite`. Puede utilizar esta opción para establecer la respuesta para el mensaje de actualización de IDT. • Se ha añadido el parámetro `--update-managed-policy` al comando `run-suite`. Puede utilizar esta opción para establecer la respuesta para el mensaje de actualización de la política administrada. • Mejoras internas y

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				correcciones de errores, entre los que se incluyen: • Para las actualizaciones automáticas del conjunto de pruebas, se han introducido mejoras en la actualización del archivo de configuración.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v3.0.2	FRQ_1.0.1	202002.00		• Para obtener más detalles, consulte el archivo CHANGELOG.md en GitHub. • Añade una actualización automática de los conjuntos de pruebas en IDT. IDT ahora puede descargar los conjuntos de pruebas más recientes disponibles para su versión de FreeRTOS. Con esta característica, puede: • Descargar los conjuntos de pruebas más recientes utilizando

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				el comando `upgrade-test-suite` . • Descargar los conjuntos de pruebas más recientes estableciendo un indicador cuando se inicie IDT. Utilice la opción `-u` *`flag`* donde la *`marca`* puede ser “y” para descargar siempre o “n” para usar la versión existente. Cuando hay varias versiones del conjunto

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				de pruebas, se utiliza la versión más reciente a menos que especifique un identificador de conjunto de pruebas al iniciar IDT. • Utilice la nueva opción `list-supported-versions` para ver las versiones del conjunto de pruebas y FreeRTOS compatibles con la versión instalada de IDT. • Vea los casos de prueba de un grupo y ejecute

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				pruebas individuales. Los conjuntos de pruebas se versionan mediante el formato `major.minor.patch`, comenzando desde 1.0.0. • Añade el comando `list-supported-products` - Enumera las versiones del conjunto de pruebas y FreeRTOS compatibles con la versión instalada de IDT. • Añade el comando `list-test-cases` - Enumera los casos de prueba que están disponibl

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				es en un grupo de pruebas. • Añade la opción `test-id` para el comando `run-suite` - Utilice esta opción para ejecutar casos de prueba individuales en un grupo de pruebas.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v1.7.1	FRQ_1.0.0	202002.00		• Para obtener más detalles, consulte el archivo CHANGELOG.md en GitHub. • Admite el método de firma de código personalizado para casos de prueba integrales inalámbricos (OTA) para que pueda utilizar sus propios comandos de firma de código y scripts para firmar cargas útiles de OTA. • Agrega una comprobación previa de los puertos serie antes del comienzo de

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				las pruebas. Las pruebas producirán rápidamente mensajes de error mejorados si el puerto serie está mal configurado en el archivo `device.json` . • Se ha agregado una AWSpolítica administrada de `AWSIoTDeviceTesterForFreeRTOSFullAccess` con permisos necesarios para ejecutar AWS IoT Device Tester. Si las nuevas versiones requieren permisos adicionales,

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				los agregamos a esta política administrada para que no tenga que actualizar los permisos de IAM. • El archivo denominado `AFQ_Report.xml` en el directorio de resultados se llama ahora `FRQ_Report.xml` .

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v1.6.2	FRQ_1.0.0	202002.00		• Admite pruebas opcionales para OTA a través de HTTPS para calificar sus placas de desarrollo de FreeRTOS. • Admite el punto de enlace ATS de AWS IoT en las pruebas. • Admite la capacidad de informar a los usuarios sobre la última versión de IDT antes del inicio del conjunto de pruebas.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v1.5.2	FRQ_1.0.0	201910.00		• Admite la calificación de dispositivos de Amazon FreeRTOS con elemento seguro (clave integrada). • Admite la portabilidad de servidores de eco configurables para grupos de pruebas de sockets seguros y wifi. • Admite el indicador multiplicador de tiempo de espera para aumentar los tiempos de espera, lo que resulta útil a la hora de solucionar problemas de errores relacionados

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
				con el tiempo de espera. • Se ha añadido una corrección de errores para el análisis de registros. • Admite el punto de enlace ats de IoT en las pruebas.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v1.4.1	FRQ_1.0.0	201908.00		• Se ha incorporado la compatibilidad con las nuevas actualizaciones de los casos de prueba y las bibliotecas de PKCS11. • Se han introducido códigos de error procesables. Para obtener más información, consulte Códigos de error de IDT • Se ha actualizado la política de IAM utilizada para ejecutar IDT.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT v1.3.2	FRQ_1.0.0	201906.00		• Se ha agregado compatibilidad para realizar pruebas con el Bluetooth de bajo consumo (BLE). • Se ha mejorado la experiencia de usuario para los comandos de la interfaz de línea de comandos (CLI) de IDT. • Se ha actualizado la política de IAM utilizada para ejecutar IDT.
IDT-FreeRTOS v1.2	FRQ_1.0.0	• FreeRTOS v1.4.8 • FreeRTOS v1.4.9		Se ha agregado compatibilidad para probar dispositivos de FreeRTOS con el sistema de creación CMAKE.

Versión AWS IoT Device Tester	Versiones del conjunto de pruebas	Versiones de FreeRTOS compatibles	Fecha de publicación	Notas de la versión
IDT-FreeRTOS v1.1	FRQ_1.0.0			
IDT-FreeRTOS v1.0	FRQ_1.0.0			

Descarga de IDT para FreeRTOS

En este tema se describen las opciones para descargar IDT para FreeRTOS. Puede utilizar uno de los siguientes enlaces de descarga de software o seguir las instrucciones para descargar IDT mediante programación.

> **Important**
>
> A partir de octubre de 2022, AWS IoT Device Tester para la calificación de AWS IoT para FreeRTOS (FRQ) 1.0 no genera informes de calificación firmados. No puede calificar nuevos dispositivos de FreeRTOS de AWS IoT para incluirlos en el Catálogo de dispositivos de socios de AWS a través del Programa de calificación de dispositivos de AWS con las versiones IDT FRQ 1.0. Si bien no puede calificar los dispositivos FreeRTOS con IDT FRQ 1.0, puede seguir probando los dispositivos FreeRTOS con FRQ 1.0. Le recomendamos que utilice IDT FRQ 2.0 para calificar y enumerar los dispositivos FreeRTOS en el Catálogo de dispositivos de socios de AWS.

Temas

- Descarga de IDT manualmente
- Descarga de IDT mediante programación

Al descargar el software, acepta el contrato de licencia de AWS IoT Device Tester incluido en el archivo de descarga.

> **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.

Descarga de IDT manualmente

En este tema se muestran las versiones compatibles de IDT para FreeRTOS. Como práctica recomendada, le recomendamos que utilice la versión más reciente de AWS IoT Device Tester que sea compatible con la versión de destino de FreeRTOS. Las nuevas versiones de FreeRTOS podrían requerir la descarga de una nueva versión de AWS IoT Device Tester. Recibirá una notificación cuando inicie una ejecución de prueba si AWS IoT Device Tester no es compatible con la versión de FreeRTOS que está utilizando.

Consulte Versiones compatibles de AWS IoT Device Tester para FreeRTOS.

Descarga de IDT mediante programación

IDT proporciona una operación de API que puede utilizar para recuperar una URL desde la que descargar IDT mediante programación. También puede usar esta operación de API para comprobar si tiene la última versión de IDT. Esta operación de API tiene el siguiente punto de conexión.

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

Para llamar a esta operación de API, debe tener el permiso para realizar la acción **`iot-device-tester:LatestIdt`**. Incluya su firma de AWS, con `iot-device-tester` como nombre del servicio.

Solicitud de API

HostOS - El sistema operativo de la máquina host. Puede elegir entre las siguientes opciones:

- `mac`
- `linux`
- `windows`

TestSuiteType - El tipo de conjunto de pruebas. Elija la opción siguiente:

`FR` - IDT para FreeRTOS

Versión del producto

(Opcional) La versión de FreeRTOS. El servicio devuelve la última versión compatible de IDT para esa versión de FreeRTOS. Si no especifica esta opción, el servicio devuelve la última versión de IDT.

Respuesta de la API

La respuesta de la API tiene el siguiente formato. `DownloadURL` incluye un archivo zip.

```
{
    "Success": True or False,
    "Message": Message,
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
 }
```

Ejemplos

Puede hacer referencia a los siguientes ejemplos para descargar IDT mediante programación. En estos ejemplos se utilizan las credenciales que almacena en las variables de entorno `AWS_ACCESS_KEY_ID` y `AWS_SECRET_ACCESS_KEY`. Para seguir las mejores prácticas recomendadas, no almacene las credenciales en el código.

Example

Ejemplo: descarga con cURL 7.75.0 o posterior (Mac y Linux)

Si tiene la versión 7.75.0 o posterior de cURL, puede usar el indicador `aws-sigv4` para firmar la solicitud de API. En este ejemplo se usa jq para analizar la URL de descarga de la respuesta.

> **Warning**
>
> El indicador `aws-sigv4` requiere que los parámetros de consulta de la solicitud GET de curl estén en el orden HostOs/ProductVersion/TestSuiteType o HostOs/TestSuiteType. Los

órdenes que no se ajusten, provocarán un error al obtener firmas no coincidentes para la cadena canónica de la puerta de enlace de la API.
Si se incluye el parámetro opcional ProductVersion, debe usar una versión de producto compatible, tal como se describe en Versiones compatibles de AWS IoT Device Tester para qFreeRTOS.

- Sustituya *us-west-2* por su Región de AWS. Para obtener la lista de códigos de región, consulte Puntos de conexión regionales.
- Sustituya *linux* por el sistema operativo de su máquina host.
- Sustituya *202107.00* por su versión de FreeRTOS.

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=202107.00&TestSuiteType=FR" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Example

Ejemplo: descarga con una versión anterior de cURL (Mac y Linux)

Puede usar el siguiente comando cURL con una firma de AWS que firme y calcule. Para obtener más información sobre cómo firmar y calcular una firma de AWS, consulte Firma de solicitudes de API de AWS.

- Sustituya *linux* por el sistema operativo de su máquina host.
- Sustituya *Timestamp* por la fecha y la hora, por ejemplo **20220210T004606Z**.
- Sustituya *Date* por la fecha, por ejemplo **20220210**.
- Sustituya *AWSRegion* por su Región de AWS. Para obtener la lista de códigos de región, consulte Puntos de conexión regionales.
- Sustituya *AWSSignature* por la firma de AWS que haya generado.

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=FR' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example

Ejemplo: descarga mediante un script de Python

En este ejemplo se utiliza la biblioteca de solicitudes de Python. Este ejemplo está adaptado del ejemplo de Python para firmar una solicitud de API de AWS en la Referencia general de AWS.

- Sustituya *us-west-2* por su región. Para obtener la lista de códigos de región, consulte Puntos de conexión regionales.
- Sustituya *linux* por el sistema operativo de su máquina host.

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
```

```
request_parameters = 'HostOs=linux&TestSuiteType=FR'

# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
```

```
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ************* SEND THE REQUEST *************
request_url = endpoint + '?' + canonical_querystring
```

```
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

Uso de IDT con el conjunto de calificación FreeRTOS 2.0 (FRQ 2.0)

El conjunto de calificación FreeRTOS 2.0 es una versión actualizada del conjunto de calificación FreeRTOS. Recomendamos a los desarrolladores que utilicen FRQ 2.0 porque está formado por casos de prueba relevantes para calificar los dispositivos que ejecutan las bibliotecas de soporte a largo plazo (LTS) de FreeRTOS.

IDT para FreeRTOS verifica el puerto de FreeRTOS del microcontrolador y si se comunica eficazmente con AWS IoT. En concreto, verifica las interfaces de la capa de portabilidad de las bibliotecas de FreeRTOS y si los respositorios de pruebas de FreeRTOS se implementan correctamente. También realiza pruebas integrales con AWS IoT Core. Las pruebas ejecutadas por IDT para FreeRTOS se definen en el repositorio de GitHub de FreeRTOS.

IDT para FreeRTOS ejecuta las pruebas como aplicaciones integradas que se instalan en el dispositivo microcontrolador que se está probando. Las imágenes binarias de la aplicación incluyen FreeRTOS, las interfaces de FreeRTOS transferidas y los controladores de dispositivos de la placa. El objetivo de las pruebas es verificar el correcto funcionamiento de las interfaces de FreeRTOS transferidas sobre los controladores de dispositivos.

IDT para FreeRTOS genera informes de pruebas que puede enviar a AWS IoT para añadir su hardware al Catálogo de dispositivos de socios de AWS. Para obtener más información, consulte el Programa de cualificación de dispositivos de AWS.

IDT para FreeRTOS se ejecuta en un equipo host (Windows, macOS o Linux) que esté conectado al dispositivo que se está probando. IDT configura y orquesta los casos de prueba y agrega los resultados. También proporciona una interfaz de línea de comandos para administrar la ejecución de las pruebas.

Para probar su dispositivo, IDT para FreeRTOS crea recursos , como objetos de AWS IoT, grupos de FreeRTOS o funciones de Lambda. Para crear estos recursos, IDT para FreeRTOS utiliza las credenciales de AWS configuradas en `config.json` para realizar llamadas a la API en su nombre. Estos recursos se aprovisionarán en distintos momentos durante una prueba.

Cuando se ejecuta IDT para FreeRTOS en su equipo host, realiza los siguientes pasos:

1. Carga y valida su dispositivo y la configuración 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 la placa supera las pruebas necesarias para la cualificación.

Temas

Requisitos previos

En esta sección se describen los requisitos previos para probar los microcontroladores con AWS IoT Device Tester.

Preparación para la calificación de FreeRTOS

> **Note**
>
> AWS IoT Device Tester para FreeRTOS recomienda encarecidamente utilizar el último parche de la versión más reciente de FreeRTOS-LTS.

IDT para FRQ 2.0 es una calificación para FreeRTOS. Antes de ejecutar IDT FRQ 2.0 para la calificación, debe completar el proceso de Calificación de su placa que se indica en la Guía de

calificación de FreeRTOS. Para realizar la portabilidad de bibliotecas, probarlas y configurar `manifest.yml`, consulte Portabilidad de las bibliotecas de FreeRTOS en la Guía de portabilidad de FreeRTOS. FRQ 2.0 contiene un proceso diferente para la calificación. Consulte Últimos cambios en la calificación en la Guía de calificación de FreeRTOS para obtener más información.

El repositorio FreeRTOS-Libraries-Integration-Tests debe estar presente para que IDT se ejecute. Consulte el archivo README.md para obtener información cómo clonar y realizar la portabilidad de este repositorio a su proyecto fuente. FreeRTOS-Libraries-Integration-Tests debe incluir `manifest.yml`, que se encuentra en la raíz de su proyecto, para que IDT se ejecute.

> **Note**
>
> IDT depende de la implementación de `UNITY_OUTPUT_CHAR` del repositorio de pruebas. Los registros de salida de la prueba y los registros del dispositivo no deben intercalarse entre sí. Consulte Implementación de macros de registro de bibliotecas en la Guía de portabilidad de FreeRTOS para obtener más detalles.

Descarga de IDT para FreeRTOS

Cada versión de FreeRTOS tiene una versión correspondiente de IDT para FreeRTOS para realizar pruebas de calificación. Descargue la versión adecuada de IDT para FreeRTOS desde Versiones compatibles de AWS IoT Device Tester para FreeRTOS.

Extraiga IDT para FreeRTOS en una ubicación del sistema de archivos en la que tenga permisos de lectura y escritura. Dado que Microsoft Windows tiene un límite de caracteres para la longitud de la ruta de acceso, extraiga IDT para FreeRTOS en un directorio raíz, como `C:\` o `D:\`.

> **Note**
>
> Varios usuarios no pueden ejecutar IDT desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Esto dará lugar a bloqueos o daños en los datos. Le recomendamos que extraiga el paquete IDT a una unidad local.

Descarga de Git

IDT debe tener Git instalado como requisito previo para garantizar la integridad del código fuente.

Para instalar Git, siga las instrucciones que se indican en la guía de GitHub. Para verificar la versión actual de Git instalada, introduzca el comando `git --version` en el terminal.

> **Warning**
>
> IDT usa Git para alinearse con el estado de un directorio, limpio o sucio. Si Git no está instalado, los grupos de pruebas de `FreeRTOSIntegrity` fallarán o no se ejecutarán como se espera. Si IDT devuelve un error, como `git executable not found` o `git command not found`, instale o vuelva a instalar Git e inténtelo de nuevo.

Creación y configuración de una cuenta de AWS

> **Note**
>
> El conjunto completo de calificaciones de IDT solo es compatible con las siguientes Regiones de AWS
>
> - Este de EE. UU. (Norte de Virginia)
> - Oeste de EE. UU. (Oregón)
> - Asia-Pacífico (Tokio)
> - Europa (Irlanda)

Para probar su dispositivo, IDT para FreeRTOS crea recursos, como objetos de AWS IoT, grupos de FreeRTOS o funciones de Lambda. Para crear esos recursos, IDT para FreeRTOS requiere que cree y configure una cuenta de AWS y una política de IAM que conceda permiso a IDT para FreeRTOS para acceder a los recursos en su nombre al ejecutar las pruebas.

Siga estos pasos para crear y configurar una cuenta de AWS.

1. Si ya dispone de una cuenta de AWS, salte al siguiente paso. O bien, cree una cuenta de AWS.
2. Siga los pasos que se indican en Creación de roles de IAM. No añada permisos ni políticas en este momento.
3. Para ejecutar pruebas de calificación OTA, vaya al paso 4. De lo contrario, vaya al paso 5.
4. Asocie la política en línea de permisos de IAM de OTA a su rol de IAM.

a.

> **Important**
>
> La siguiente plantilla de política concede permiso a IDT para crear roles, crear políticas y asociar políticas a roles. IDT para FreeRTOS utiliza estos permisos para las pruebas que crean roles. Aunque la plantilla de política no proporciona privilegios de administrador al usuario, los permisos se pueden utilizar para obtener acceso de administrador a su cuenta de AWS.

b. Siga estos pasos para asociar los permisos necesarios a su rol de IAM:

   i. En la pestaña Permisos, seleccione Añadir permisos.

   ii. Elija Crear política insertada.

   iii. Elija la pestaña JSON y copie los siguientes permisos en el cuadro de texto JSON. Utilice la plantilla que aparece en La mayoría de las regiones si no se encuentra en la región de China. Si se encuentra en la región de China, utilice la plantilla que aparece en Regiones de Pekín y Ningxia.

   Most Regions

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": [
                "arn:aws:iotdeviceadvisor:*:*:suiterun/*/*",
                "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService":
 "iotdeviceadvisor.amazonaws.com"
                }
            }
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke*",
                "iam:ListRoles",
                "iot:Connect",
                "iot:CreateJob",
                "iot:DeleteJob",
                "iot:DescribeCertificate",
                "iot:DescribeEndpoint",
                "iot:DescribeJobExecution",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:DeleteLogGroup",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iot/
deviceadvisor/*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": "logs:GetLogEvents",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iot/
deviceadvisor/*:log-stream:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:policy/idt*",
                "arn:aws:iam::*:role/idt*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:*::parameter/aws/service/ami-amazon-linux-
latest/amzn2-ami-hvm-x86_64-gp2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
            "Resource": [
                "*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:CreateKeyPair",
                "ec2:DeleteKeyPair"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:key-pair/idt-ec2-ssh-key-*"
            ]
        },
        {
            "Effect": "Allow",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "aws:ResourceTag/Owner": "IoTDeviceTester"
                }
            },
            "Action": [
                "ec2:TerminateInstances",
                "ec2:DeleteSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Beijing and Ningxia Regions

La siguiente plantilla de política se puede utilizar en las regiones de Pekín y Ningxia.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
```

```
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws-cn:iam::*:policy/idt*",
                "arn:aws-cn:iam::*:role/idt*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws-cn:ssm:*::parameter/aws/service/ami-amazon-
linux-latest/amzn2-ami-hvm-x86_64-gp2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateKeyPair",
                "ec2:DeleteKeyPair"
            ],
            "Resource": [
                "arn:aws-cn:ec2:*:*:key-pair/idt-ec2-ssh-key-*"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "aws-cn:ResourceTag/Owner": "IoTDeviceTester"
                }
            },
            "Action": [
                "ec2:TerminateInstances",
                "ec2:DeleteSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

iv. Cuando haya terminado, elija Review policy (Revisar política).

v. Introduzca IDTFreeRTOSIAMPermissions como nombre para la política.

vi. Elija Create Policy (Crear política).

5. Asocie AWSIoTDeviceTesterForFreeRTOSFullAccess a su rol de IAM.

   a. Para asociar los permisos necesarios a su rol de IAM:

      i. En la pestaña Permisos, seleccione Añadir permisos.

      ii. Seleccione Attach policies (Asociar políticas).

      iii. Busque la política AWSIoTDeviceTesterForfreeRTOSFullAccess. Marque la casilla.

   b. Elija Add permissions (Agregar permisos).

6. Exporte las credenciales para IDT. Consulte Obtención de credenciales de rol de IAM para el acceso a la CLI para obtener más información.

Política administrada de AWS IoT Device Tester

La política administrada `AWSIoTDeviceTesterForFreeRTOSFullAccess` contiene los siguientes permisos de AWS IoT Device Tester para la verificación de versiones, las características de actualización automática y la recopilación de métricas.

- `iot-device-tester:SupportedVersion`

Otorga a AWS IoT Device Tester el permiso para obtener la lista de productos, conjuntos de pruebas y versiones de IDT compatibles.

- `iot-device-tester:LatestIdt`

  Otorga a AWS IoT Device Tester el permiso para obtener la versión de IDT más reciente que se puede descargar.

- `iot-device-tester:CheckVersion`

  Otorga a AWS IoT Device Tester el permiso para comprobar la compatibilidad de las versiones de IDT, los conjuntos de pruebas y los productos.

- `iot-device-tester:DownloadTestSuite`

  Otorga a AWS IoT Device Tester el permiso para descargar conjuntos de pruebas.

- `iot-device-tester:SendMetrics`

  Otorga a AWS el permiso para recopilar métricas sobre el uso interno de AWS IoT Device Tester.

(Opcional) Instale AWS Command Line Interface

Tal vez prefiera utilizar la AWS CLI para realizar algunas operaciones. Si no tiene la AWS CLI instalada, siga las instrucciones que se indican en Instalación de la AWS CLI.

Configure la AWS CLI para la región de AWS que desea utilizar ejecutando aws configure desde una línea de comandos. Para obtener más información sobre las regiones de AWS que admiten IDT para FreeRTOS, consulte Regiones de AWS y puntos de conexión. Para obtener más información sobre aws configure, consulte Configuración rápida con aws configure.

Preparación para probar su placa de microcontrolador por primera vez

Puede usar IDT para FreeRTOS para probar la implementación de las bibliotecas FreeRTOS. Una vez que haya realizado la portabilidad las bibliotecas de FreeRTOS para los controladores de dispositivos de su placa, utilice AWS IoT Device Tester para ejecutar las pruebas de calificación en su placa del microcontrolador.

Adición de capas de portabilidad de bibliotecas e implementación de un repositorio de pruebas de FreeRTOS

Para realizar la portabilidad de FreeRTOS para su dispositivo, consulte la Guía de portabilidad de FreeRTOS. Al implementar el repositorio de pruebas de FreeRTOS y realizar la portabilidad de las capas de FreeRTOS, debe proporcionar un `manifest.yml` con las rutas a cada biblioteca, incluido el repositorio de pruebas. Este archivo se ubicará en el directorio raíz de su código fuente. Consulte las instrucciones del archivo de manifiesto para obtener más información.

Configuración de sus credenciales de AWS

Debe configurar las credenciales de AWS para AWS IoT Device Tester para comunicarse con la nube de AWS. Para obtener más información, consulte Configuración de credenciales y regiones de AWS para desarrollo. Las credenciales de AWS válidas se especifican en el archivo de configuración *devicetester_extract_location*/devicetester_freertos_*[win|mac|linux]*/configs/config.json.

```
"auth": {
   "method": "environment"
}

"auth": {
    "method": "file",
    "credentials": {
        "profile": "<your-aws-profile>"
    }
}
```

El atributo `auth` del archivo `config.json` tiene un campo de método que controla la autenticación de AWS y se puede declarar como archivo o entorno. Al configurar el campo como entorno, se extraen las credenciales de AWS de las variables de entorno de la máquina host. Al configurar el campo en un archivo, se importa un perfil específico del archivo de configuración `.aws/credentials`.

Creación de un grupo de dispositivos en IDT para FreeRTOS

Los dispositivos que se vayan a probar se organizan en grupos de dispositivos. Cada grupo de dispositivos se compone de uno o varios dispositivos idénticos. Puede configurar IDT para FreeRTOS para probar un solo dispositivo o varios dispositivos de un grupo. Para acelerar el

proceso de calificación, IDT para FreeRTOS puede probar en paralelo dispositivos con la misma especificación. Utiliza un método de turnos rotativos para ejecutar un grupo de pruebas diferentes en todos los dispositivos de un grupo de dispositivos.

El archivo `device.json` tiene una matriz en su nivel superior. Cada atributo de la matriz es un nuevo grupo de dispositivos. Cada grupo de dispositivos tiene un atributo de matriz de dispositivos, que tiene varios dispositivos declarados. En la plantilla, hay un grupo de dispositivos y solo un dispositivo en ese grupo de dispositivos. Puede añadir uno o varios dispositivos a un grupo de dispositivos editando la sección `devices` de la plantilla `device.json` en la carpeta `configs`.

> **Note**
>
> Todos los dispositivos del mismo grupo deben tener la misma especificación técnica y SKU. Para habilitar las compilaciones en paralelo del código fuente para diferentes grupos de prueba, IDT para FreeRTOS copia el código fuente en una carpeta de resultados dentro de la carpeta extraída de IDT para FreeRTOS. Debe hacer referencia a la ruta del código fuente en el comando build o flash con la variable `testdata.sourcePath`. IDT para FreeRTOS reemplaza esta variable con una ruta temporal del código fuente copiado. Para obtener más información, consulte Variables de IDT para FreeRTOS.

A continuación se muestra un ejemplo de un archivo `device.json` utilizado para crear un grupo de dispositivos con varios dispositivos.

```
[
    {
        "id": "pool-id",
        "sku": "sku",
        "features": [
           {
              "name": "Wifi",
              "value": "Yes | No"
           },
           {
              "name": "Cellular",
              "value": "Yes | No"
           },
           {
              "name": "BLE",
              "value": "Yes | No"
```

```
          },
          {
             "name": "PKCS11",
             "value": "RSA | ECC | Both"
          },
          {
              "name": "OTA",
              "value": "Yes | No",
              "configs": [
              {
                  "name": "OTADataPlaneProtocol",
                  "value": "MQTT | HTTP | None"
              }
            ]
          },
          {
             "name": "KeyProvisioning",
             "value": "Onboard | Import | Both | No"
          }
        ],
        "devices": [
          {
            "id": "device-id",
            "connectivity": {
              "protocol": "uart",
              "serialPort": "/dev/tty*"
            },
            "secureElementConfig" : {
              "publicKeyAsciiHexFilePath": "absolute-path-to/public-key-txt-file:
 contains-the-hex-bytes-public-key-extracted-from-onboard-private-key",
              "publiDeviceCertificateArn": "arn:partition:iot:region:account-
id:resourcetype:resource:qualifier",
              "secureElementSerialNumber": "secure-element-serialNo-value",
              "preProvisioned"           : "Yes | No",
              "pkcs11JITPCodeVerifyRootCertSupport": "Yes | No"
            },
            "identifiers": [
              {
                "name": "serialNo",
                "value": "serialNo-value"
              }
            ]
          }
        ]
```

```
    }
]
```

En el archivo `device.json` se utilizan los siguientes atributos:

`id`

Un ID alfanumérico definido por el usuario que identifica de manera exclusiva a un grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben ser del mismo tipo. Cuando se ejecuta un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo.

`sku`

Un valor alfanumérico que identifica de forma única la placa que está probando. El SKU se utiliza para realizar un seguimiento de placas cualificadas.

> **Note**
>
> Si desea enumerar la placa en el Catálogo de dispositivos de socios de AWS, la SKU que especifique aquí debe coincidir con la SKU que utilice en el proceso de enumeración.

`features`

Una matriz que contiene las funciones compatibles del dispositivo. AWS IoT Device Tester utiliza esta información para seleccionar las pruebas de calificación que se van a ejecutar.

Los valores admitidos son:

`Wifi`

Indica si la placa tiene capacidades Wi-Fi.

`Cellular`

Indica si la placa tiene capacidad móvil.

`PKCS11`

Indica el algoritmo de criptografía de clave pública que admite la placa. PKCS11 es necesario para la cualificación. Los valores admitidos son `ECC`, `RSA` y `Both`. `Both` indica que la placa admite `ECC` y `RSA`.

KeyProvisioning

Indica el método para escribir un certificado de cliente X.509 de confianza en la placa.

Los valores válidos son `Import`, `Onboard`, `Both` y `No`. Se requiere el aprovisionamiento de las claves `Onboard`, `Both` o `No` para la calificación. La opción `Import` por sí sola no es válida para la calificación.

- Utilice `Import` solo si su placa permite la importación de claves privadas. Seleccionar `Import` no es una configuración válida para la calificación y solo debe usarse con fines de prueba, específicamente en los casos de prueba de PKCS11. Se requiere `Onboard`, `Both` o `No` para la calificación.
- Utilice `Onboard` si la placa admite claves privadas integradas (por ejemplo, si su dispositivo tiene un elemento seguro o si prefiere generar su propio par de claves de dispositivo y certificado). Procure añadir un elemento `secureElementConfig` en cada una de las secciones del dispositivo e introduzca la ruta absoluta del archivo de claves públicas en el campo `publicKeyAsciiHexFilePath`.
- Utilice `Both` si la placa admite tanto la importación de claves privadas como la generación de claves integradas para el aprovisionamiento de claves.
- Utilice `No` si la placa no admite el aprovisionamiento de claves. `No` solo es una opción válida cuando el dispositivo también está aprovisionado previamente.

OTA

Indica si la placa admite la funcionalidad de actualización transparente (OTA). El atributo `OtaDataPlaneProtocol` indica qué protocolo de plano de datos de OTA admite el dispositivo. Para la calificación, se necesita OTA con el protocolo de plano de datos HTTP o MQTT. Para omitir la ejecución de pruebas OTA durante las pruebas, defina la característica OTA en `No` y el atributo `OtaDataPlaneProtocol` en `None`. No se tratará de una ejecución de calificación.

BLE

Indica si la placa admite Bluetooth de bajo consumo (BLE).

devices.id

Un identificador único y definido por el usuario para el dispositivo que se está probando.

devices.connectivity.serialPort

El puerto de serie del equipo host utilizado para conectarse a los dispositivos que se van a probar.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Es obligatorio si la placa no está `pre-provisioned` o si no se proporciona `PublicDeviceCertificateArn`. Dado que `Onboard` es un tipo de aprovisionamiento de claves obligatorio, este campo es obligatorio actualmente para el grupo de pruebas FullTransportInterfaceTLS. Si su dispositivo está `pre-provisioned`, `PublicKeyAsciiHexFilePath` es opcional y no es necesario incluirlo.

El bloque siguiente es una ruta absoluta al archivo que contiene la clave pública de bytes hexadecimales extraída de la clave privada `Onboard`.

```
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

Si la clave pública tiene el formato .der, puede codificar directamente la clave pública en formato hexadecimal para generar el archivo hexadecimal.

Para generar el archivo hexadecimal a partir de una clave pública .der, introduzca el siguiente comando xxd:

```
xxd -p pubkey.der > outFile
```

Si su clave pública tiene el formato .pem, puede extraer los encabezados y pies de página codificados en base64 y decodificarlos en formato binario. A continuación, realice la codificación hexadecimal de la cadena binaria para generar el archivo hexadecimal.

Para generar un archivo hexadecimal para una clave pública .pem, realice lo siguiente:

1. Ejecute el siguiente comando base64 para eliminar el encabezado y el pie de página en base64 de la clave pública. A continuación, la clave decodificada, denominada `base64key`, se envía al archivo `pubkey.der`:

   ```
   base64 –decode base64key > pubkey.der
   ```

2. Ejecute el siguiente comando xxd para convertir `pubkey.der` a formato hexadecimal. La clave resultante se guarda como *`outFile`*.

```
xxd -p pubkey.der > outFile
```

`devices.secureElementConfig.PublicDeviceCertificateArn`

El ARN del certificado del elemento seguro que se ha cargado en AWS IoT Core. Para obtener más información sobre cómo cargar el certificado en AWS IoT Core, consulte Certificados de cliente X.509 en la Guía para desarrolladores de AWS IoT.

`devices.secureElementConfig.SecureElementSerialNumber`

(Opcional) Número de serie del elemento seguro. El número de serie se puede usar para crear certificados de dispositivo para el aprovisionamiento de claves JITR.

`devices.secureElementConfig.preProvisioned`

(Opcional) Indique el valor “Sí” si el dispositivo tiene un elemento seguro aprovisionado previamente con credenciales bloqueadas, que no puede importar, crear ni destruir objetos. Si este atributo está establecido en Sí, debe proporcionar las etiquetas pkcs11 correspondientes.

`devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport`

(Opcional) Configúrelo en Sí si la implementación de corePKCS11 del dispositivo admite el almacenamiento para JITP. Esto habilitará la prueba `codeverify` de JITP al probar el PKCS 11 principal y requerirá que se proporcionen la clave de la verificación del código, el certificado JITP y las etiquetas PKCS 11 del certificado raíz.

`identifiers`

(Opcional) Una matriz de pares de nombre-valor arbitrarios. Puede utilizar estos valores en los comandos Build y Flash descritos en la siguiente sección.

Configurar ajustes de Build, Flash y Test

IDT para FreeRTOS crea e instala las pruebas en su placa automáticamente. Para habilitarlo, debe configurar IDT para que ejecute los comandos build y flash para el hardware. Los ajustes de compilación e instalación se configuran en el archivo de plantilla `config` que se encuentra en la carpeta `userdata.json`.

Configurar ajustes para probar dispositivos

Los ajustes de compilación, instalación y prueba se realizan en el archivo `configs/userdata.json`. El siguiente ejemplo de JSON muestra cómo puede configurar IDT para FreeRTOS para probar varios dispositivos:

```
{
    "sourcePath": "</path/to/freertos>",
    "retainModifiedSourceDirectories": true | false,
    "freeRTOSVersion": "<freertos-version>",
    "freeRTOSTestParamConfigPath": "{{testData.sourcePath}}/path/from/source/path/to/
test_param_config.h",
    "freeRTOSTestExecutionConfigPath": "{{testData.sourcePath}}/path/from/source/path/
to/test_execution_config.h",
    "buildTool": {
        "name": "your-build-tool-name",
        "version": "your-build-tool-version",
        "command": [
            "<build command> -any-additional-flags {{testData.sourcePath}}"
        ]
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "<flash command> -any-additional-flags {{testData.sourcePath}} -any-
additional-flags"
        ]
    },
    "testStartDelayms": 0,
    "echoServerConfiguration": {
      "keyGenerationMethod": "EC | RSA",
      "serverPort": 9000
    },
    "otaConfiguration": {
        "otaE2EFirmwarePath": "{{testData.sourcePath}}/relative-path-to/ota-image-
generated-in-build-process",
        "otaPALCertificatePath": "/path/to/ota/pal/certificate/on/device",
        "deviceFirmwarePath" : "/path/to/firmware/image/name/on/device",
        "codeSigningConfiguration": {
            "signingMethod": "AWS | Custom",
            "signerHashingAlgorithm": "SHA1 | SHA256",
            "signerSigningAlgorithm": "RSA | ECDSA",
```

```
            "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
            "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
            "signerCertificateFileName": "signerCertificate-file-name",
            "compileSignerCertificate": true | false,
            // ***********Use signerPlatform if you choose AWS for
 signingMethod***************
            "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF"

            ]
         }
    },
    **********
    This section is used for PKCS #11 labels of private key, public key, device
 certificate, code verification key, JITP certificate, and root certificate.
    When configuring PKCS11, you set up labels and you must provide the labels of the
 device certificate, public key,
    and private key for the key generation type (EC or RSA) it was created with. If
 your device supports PKCS11 storage of JITP certificate,
    code verification key, and root certificate, set
 'pkcs11JITPCodeVerifyRootCertSupport' to 'Yes' in device.json and provide the
 corresponding labels.
    **********
    "pkcs11LabelConfiguration":{
        "pkcs11LabelDevicePrivateKeyForTLS": "<device-private-key-label>",
        "pkcs11LabelDevicePublicKeyForTLS": "<device-public-key-label>",
        "pkcs11LabelDeviceCertificateForTLS": "<device-certificate-label>",
        "pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS": "<preprovisioned-ec-
device-private-key-label>",
        "pkcs11LabelPreProvisionedECDevicePublicKeyForTLS": "<preprovisioned-ec-device-
public-key-label>",
        "pkcs11LabelPreProvisionedECDeviceCertificateForTLS": "<preprovisioned-ec-
device-certificate-label>",
        "pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS": "<preprovisioned-rsa-
device-private-key-label>",
        "pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS": "<preprovisioned-rsa-
device-public-key-label>",
        "pkcs11LabelPreProvisionedRSADeviceCertificateForTLS": "<preprovisioned-rsa-
device-certificate-label>",
        "pkcs11LabelCodeVerifyKey": "<code-verification-key-label>",
        "pkcs11LabelJITPCertificate": "<JITP-certificate-label>",
        "pkcs11LabelRootCertificate": "<root-certificate-label>"
     }
```

```
}
```

A continuación se indican los atributos usados en `userdata.json`:

`sourcePath`

La ruta a la raíz del código fuente de FreeRTOS trasladado.

`retainModifiedSourceDirectories`

(Opcional) Compruebe si se deben conservar los directorios fuente modificados utilizados durante la creación e instalación con fines de depuración. Si se establece en`true`, el nombre de los directorios fuente modificados cambia a retainedSrc y se encuentran en las carpetas de registros de resultados de cada grupo de pruebas ejecutado. Si no se incluye, el campo se establece de forma predeterminada en `false`.

`freeRTOSTestParamConfigPath`

La ruta al archivo `test_param_config.h` para la integración de FreeRTOS-Libraries-Integration-Tests. Este archivo debe usar la variable de marcador de posición `{{testData.sourcePath}}` para que sea relativo a la raíz del código fuente. AWS IoT Device Tester usa los parámetros de este archivo para configurar las pruebas.

`freeRTOSTestExecutionConfigPath`

La ruta al archivo `test_execution_config.h` para la integración de FreeRTOS-Libraries-Integration-Tests. Este archivo debe usar la variable de marcador de posición `{{testData.sourcePath}}` para que sea relativo a la raíz del repositorio. AWS IoT Device Tester usa este archivo para controlar qué pruebas deben ejecutarse.pruebas.

`freeRTOSVersion`

La versión de FreeRTOS, incluida la versión del parche utilizada en su implementación. Consulte Versiones compatibles de AWS IoT Device Tester para FreeRTOS para ver las versiones de FreeRTOS compatibles con AWS IoT Device Tester para FreeRTOS.

`buildTool`

El comando para crear el código fuente. Todas las referencias a la ruta de código fuente en el comando build deben sustituirse por la variable `{{testData.sourcePath}}` de AWS IoT Device Tester. Utilice el marcador de posición `{{config.idtRootPath}}` para hacer referencia a un script de creación relativo a la ruta raíz de AWS IoT Device Tester.

`flashTool`

El comando para instalar una imagen en el dispositivo. Todas las referencias a la ruta de código fuente en el comando flash deben sustituirse por la variable de AWS IoT Device Tester `{{testData.sourcePath}}`. Utilice el marcador de posición `{{config.idtRootPath}}` para hacer referencia a un script de instalación relativo a la ruta raíz de AWS IoT Device Tester.

> **Note**
>
> La nueva estructura de pruebas de integración con FRQ 2.0 no requiere variables de ruta como `{{enableTests}}` y `{{buildImageName}}`. Las pruebas integrales OTA se ejecutan con las plantillas de configuración que se proporcionan en el repositorio GitHub de FreeRTOS-Libraries-Integration-Tests. Si los archivos del repositorio de GitHub están presentes en el proyecto fuente principal, el código fuente no cambia entre pruebas. Si necesita una imagen de creación diferente para las pruebas integrales OTA, debe crear esta imagen en el script de creación y especificarla en el archivo `userdata.json` especificado en `otaConfiguration`.

`testStartDelayms`

Especifica cuántos milisegundos esperará el ejecutor de pruebas de FreeRTOS antes de empezar a ejecutar las pruebas. Esto puede resultar útil si el dispositivo que se está probando comienza a generar información de prueba importante antes de que IDT tenga la oportunidad de conectarse y empezar a registrar datos debido a problemas de latencia de red o de otro tipo. Este valor se aplica únicamente a los grupos de pruebas de FreeRTOS y no a otros grupos de pruebas que no utilizan el ejecutor de pruebas de FreeRTOS, como las pruebas OTA. Si recibe un error de tipo se esperaban 10 elementos, pero se han recibido 5, este campo debe establecerse en 5000.

`echoServerConfiguration`

Los ajustes para configurar el servidor echo para la prueba de TLS. Este campo es obligatorio.

`keyGenerationMethod`

El servidor echo se configura con esta opción. Las opciones son EC o RSA.

`serverPort`

El número de puerto en el que se ejecuta el servidor echo.

`otaConfiguration`

La configuración para las pruebas PAL y E2E OTA. Este campo es obligatorio.

`otaE2EFirmwarePath`

Ruta a la imagen binaria OTA que IDT utiliza para las pruebas integrales OTA.

`otaPALCertificatePath`

La ruta al certificado para la prueba PAL OTA en el dispositivo. Se utiliza para verificar la firma. Por ejemplo, ecdsa-sha256-signer.crt.pem.

`deviceFirmwarePath`

La ruta al nombre con codificación rígida para el arranque de la imagen del firmware. Si el dispositivo no utiliza el sistema de archivos para el arranque del firmware, especifique este campo como `'NA'`. Si el dispositivo utiliza el sistema de archivos para el arranque del firmware, especifique la ruta o el nombre de la imagen de arranque del firmware.

`codeSigningConfiguration`

`signingMethod`

El método de firma de código. Los valores posibles son AWS o Personalizado.

> **Note**
>
> Para las regiones de Pekín y Ningxia, utilice Personalizado. La firma de código de AWS no se admite en esa región.

`signerHashingAlgorithm`

El algoritmo hash admitido en el dispositivo. Los valores posibles son `SHA1` o `SHA256`.

`signerSigningAlgorithm`

El algoritmo de firma admitido en el dispositivo. Los valores posibles son `RSA` o `ECDSA`.

`signerCertificate`

Certificado de confianza utilizado para OTA. Para el método de firma de código de AWS, utilice el Nombre de recurso de Amazon (ARN) del certificado de confianza cargado en

AWS Certificate Manager. Para el método de firma de código personalizado, utilice la ruta absoluta al archivo de certificado del firmante. Para obtener información sobre cómo crear un certificado de confianza, consulte Creación de un certificado de firma de código.

untrustedSignerCertificate

El ARN o ruta de archivo de un segundo certificado utilizado en algunas pruebas OTA como certificado que no es de confianza. Para obtener información sobre cómo crear un certificado, consulte Creación de un certificado de firma de código.

signerCertificateFileName

El nombre de la ruta del certificado de firma de código en el dispositivo. Este valor debe coincidir con el nombre de archivo que proporcionó al ejecutar el comando `aws acm import-certificate`.

compileSignerCertificate

Valor booleano que determina el estado del certificado de verificación de firma. Los valores válidos son `true` y `false`.

Establezca este valor en verdadero si el certificado de verificación de firma del firmante del código no está aprovisionado o instalado. Debe estar compilado en el proyecto. AWS IoT Device Tester recupera el certificado de confianza y lo compila en `aws_codesigner_certificate.h`.

signerPlatform

El algoritmo de firma y hash que AWS Code Signer utiliza al crear el trabajo de actualización de OTA. En la actualidad, los valores posibles para este campo son `AmazonFreeRTOS-TI-CC3220SF` y `AmazonFreeRTOS-Default`.

- Elija `AmazonFreeRTOS-TI-CC3220SF` si es `SHA1` y `RSA`.
- Elija `AmazonFreeRTOS-Default` si es `SHA256` y `ECDSA`.
- Si necesita `SHA256` | `RSA` o `SHA1` | `ECDSA` para su configuración, contacte con nosotros para obtener ayuda adicional.
- Configure `signCommand` si eligió `Custom` para `signingMethod`.

signCommand

Se necesitan dos marcadores de posición `{{inputImageFilePath}}` y `{{outputSignatureFilePath}}` en el comando. `{{inputImageFilePath}}`

es la ruta del archivo de la imagen creada por IDT que se va a firmar. `{{outputSignatureFilePath}}` es la ruta del archivo de la firma que generará el script.

`pkcs11LabelConfiguration`

La configuración de etiquetas PKCS11 requiere al menos un conjunto de etiquetas: etiqueta de certificado de dispositivo, etiqueta de clave pública y etiqueta de clave privada para ejecutar los grupos de prueba de PKCS11. Las etiquetas PKCS11 necesarias se basan en la configuración del dispositivo en el archivo `device.json`. Si el aprovisionamiento previo está establecido en Sí en `device.json`, las etiquetas necesarias deben ser una de las siguientes, en función de lo que se elija para la función PKCS11.

- `PreProvisionedEC`
- `PreProvisionedRSA`

Si el aprovisionamiento previo está establecido en No en `device.json`, las etiquetas necesarias son:

- `pkcs11LabelDevicePrivateKeyForTLS`
- `pkcs11LabelDevicePublicKeyForTLS`
- `pkcs11LabelDeviceCertificateForTLS`

Las tres etiquetas siguientes son necesarias solo si selecciona Sí para `pkcs11JITPCodeVerifyRootCertSupport` en el archivo`device.json`.

- `pkcs11LabelCodeVerifyKey`
- `pkcs11LabelRootCertificate`
- `pkcs11LabelJITPCertificate`

Los valores de estos campos deben coincidir con los valores definidos en la Guía de portabilidad de FreeRTOS.

`pkcs11LabelDevicePrivateKeyForTLS`

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 de la clave privada. En el caso de los dispositivos con soporte integrado y de importación para el aprovisionamiento de claves, esta etiqueta se utiliza para realizar pruebas. Esta etiqueta puede ser diferente de la definida para el caso de aprovisionamiento previo. Si el aprovisionamiento de claves está establecido en No y el aprovisionamiento previo en Sí, en `device.json`, no estará definido.

pkcs11LabelDevicePublicKeyForTLS

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 de la clave pública. En el caso de los dispositivos con soporte integrado y de importación para el aprovisionamiento de claves, esta etiqueta se utiliza para realizar pruebas. Esta etiqueta puede ser diferente de la definida para el caso de aprovisionamiento previo. Si el aprovisionamiento de claves está establecido en No y el aprovisionamiento previo en Sí, en `device.json`, no estará definido.

pkcs11LabelDeviceCertificateForTLS

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 del certificado del dispositivo. En el caso de los dispositivos con soporte integrado y de importación para el aprovisionamiento de claves, esta etiqueta se utilizará para realizar pruebas. Esta etiqueta puede ser diferente de la definida para el caso de aprovisionamiento previo. Si el aprovisionamiento de claves está establecido en No y el aprovisionamiento previo en Sí, en `device.json`, no estará definido.

pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 de la clave privada. En el caso de dispositivos con elementos seguros o limitaciones de hardware, esta tendrá una etiqueta diferente para conservar las credenciales de AWS IoT. Si su dispositivo admite el aprovisionamiento previo con una clave EC, proporcione esta etiqueta. Si el aprovisionamiento previo está establecido en Sí en `device.json`, se debe indicar esta etiqueta, `pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS` o ambas. Esta etiqueta puede ser diferente de la definida para los casos de integración e importación.

pkcs11LabelPreProvisionedECDevicePublicKeyForTLS

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 de la clave pública. En el caso de dispositivos con elementos seguros o limitaciones de hardware, esta tendrá una etiqueta diferente para conservar las credenciales de AWS IoT. Si su dispositivo admite el aprovisionamiento previo con una clave EC, proporcione esta etiqueta. Si el aprovisionamiento previo está establecido en Sí en `device.json`, se debe indicar esta etiqueta, `pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS` o ambas. Esta etiqueta puede ser diferente de la definida para los casos de integración e importación.

pkcs11LabelPreProvisionedECDeviceCertificateForTLS

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 del certificado del dispositivo. En el caso de dispositivos con elementos seguros o limitaciones de hardware, esta tendrá una etiqueta diferente para conservar las credenciales de AWS IoT. Si su dispositivo admite el aprovisionamiento previo con una clave EC, proporcione esta etiqueta. Si el

aprovisionamiento previo está establecido en Sí en `device.json`, se debe indicar esta etiqueta, `pkcs11LabelPreProvisionedRSADeviceCertificateForTLS` o ambas. Esta etiqueta puede ser diferente de la definida para los casos de integración e importación.

`pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS`

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 de la clave privada. En el caso de dispositivos con elementos seguros o limitaciones de hardware, esta tendrá una etiqueta diferente para conservar las credenciales de AWS IoT. Si su dispositivo admite el aprovisionamiento previo con una clave RSA, proporcione esta etiqueta. Si el aprovisionamiento previo está establecido en Sí en `device.json`, se debe indicar esta etiqueta, `pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS` o ambas.

`pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS`

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 de la clave pública. En el caso de dispositivos con elementos seguros o limitaciones de hardware, esta tendrá una etiqueta diferente para conservar las credenciales de AWS IoT. Si su dispositivo admite el aprovisionamiento previo con una clave RSA, proporcione esta etiqueta. Si el aprovisionamiento previo está establecido en Sí en `device.json`, se debe indicar esta etiqueta, `pkcs11LabelPreProvisionedECDevicePublicKeyForTLS` o ambas.

`pkcs11LabelPreProvisionedRSADeviceCertificateForTLS`

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 del certificado del dispositivo. En el caso de dispositivos con elementos seguros o limitaciones de hardware, esta tendrá una etiqueta diferente para conservar las credenciales de AWS IoT. Si su dispositivo admite el aprovisionamiento previo con una clave RSA, proporcione esta etiqueta. Si el aprovisionamiento previo está establecido en Sí en `device.json`, se debe indicar esta etiqueta, `pkcs11LabelPreProvisionedECDeviceCertificateForTLS` o ambas.

`pkcs11LabelCodeVerifyKey`

(Opcional) Esta etiqueta se utiliza como etiqueta PKCS 11 de la clave de verificación del código. Si su dispositivo admite el almacenamiento PKCS 11 del certificado JITP, la clave de verificación de código y el certificado raíz, proporcione esta etiqueta. Si `pkcs11JITPCodeVerifyRootCertSupport` en `device.json` está establecido en Sí, se debe indicar esta etiqueta.

`pkcs11LabelJITPCertificate`

(Opcional) Esta etiqueta se usa para la etiqueta PKCS 11 del certificado JITP. Si su dispositivo admite el almacenamiento PKCS 11 del certificado JITP, la clave de verificación de código y el

certificado raíz, proporcione esta etiqueta. Si `pkcs11JITPCodeVerifyRootCertSupport` en `device.json` está establecido en Sí, se debe indicar esta etiqueta.

Variables de IDT para FreeRTOS

Los comandos para compilar el código y actualizar el dispositivo podrían requerir conectividad u otro tipo de información sobre sus dispositivos para ejecutarse correctamente. AWS IoT Device Tester le permite hacer referencia a la información del dispositivo en los comandos flash y build con JsonPath. Al utilizar expresiones JsonPath sencillas, puede recuperar la información necesaria tal y como se especifica en su archivo `device.json`.

Variables de ruta

IDT para FreeRTOS define las siguientes variables de ruta que se pueden utilizar en líneas de comandos y archivos de configuración:

`{{testData.sourcePath}}`

Amplía la ruta del código fuente. Si utiliza esta variable, se debe utilizar tanto en los comandos flash como en los comandos build.

`{{device.connectivity.serialPort}}`

Amplía al puerto serie.

`{{device.identifiers[?(@.name == 'serialNo')].value[0]}}`

Se amplía hasta el número de serie de su dispositivo.

`{{config.idtRootPath}}`

Se expande hasta la ruta raíz AWS IoT Device Tester.

Uso de la interfaz de usuario de IDT para FreeRTOS para ejecutar el conjunto de calificación FreeRTOS 2.0 (FRQ 2.0)

AWS IoT Device Tester para FreeRTOS (IDT para FreeRTOS) incluye una interfaz de usuario (IU) basada en web en la que puede interactuar con la aplicación de línea de comandos de IDT y los archivos de configuración relacionados. Utilice la IU de IDT para FreeRTOS para crear una nueva configuración o modificar una existente para el dispositivo. También puede usar la IU para llamar a la aplicación de IDT y ejecutar las pruebas de FreeRTOS en su dispositivo.

Para obtener información sobre cómo utilizar la línea de comandos para ejecutar pruebas de calificación, consulte Preparación para probar su placa de microcontrolador por primera vez.

En esta sección se describen los requisitos previos de la IU de IDT para FreeRTOS y cómo ejecutar las pruebas de calificación desde la IU.

Temas

Requisitos previos

Para ejecutar pruebas a través de la IU de AWS IoT Device Tester (IDT) para FreeRTOS, debe cumplir los requisitos previos de la página Requisitos previos para la calificación de IDT para FreeRTOS (FRQ) 2.x.

Configurar credenciales de AWS

Debe configurar sus credenciales de usuario de IAM para el usuario de AWS creado en Creación y configuración de una cuenta de AWS. Puede especificar sus credenciales de una de las dos formas siguientes:

- En un archivo de credenciales
- Como 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 y 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
```

> **Note**
>
> Si no usa el perfil `default` de AWS, debe especificar el nombre del perfil en la IU de IDT para FreeRTOS. Para obtener más información sobre los perfiles, 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. La IU de IDT para FreeRTOS utiliza 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
```

Apertura de la IU de IDT para FreeRTOS

Para abrir la IU de IDT para FreeRTOS

1. Descargue una versión compatible de IDT para FreeRTOS. A continuación, extraiga el archivo descargado a un directorio para el que tenga permisos de lectura y escritura.
2. Vaya al directorio de instalación de IDT para FreeRTOS:

```
cd devicetester-extract-location/bin
```

3. Ejecute el siguiente comando para abrir la IU de IDT para FreeRTOS:

   Linux

   ```
   .devicetester_ui_linux_x86-64
   ```

   Windows

   ```
   ./devicetester_ui_win_x64-64
   ```

   macOS

   ```
   ./devicetester_ui_mac_x86-64
   ```

   > **Note**
   >
   > En macOS, para permitir que tu sistema ejecute la IU, vaya a Preferencias del sistema -> Seguridad y privacidad. Cuando ejecute las pruebas, es posible que tenga que hacerlo tres veces más.

   La IU de IDT para FreeRTOS se abre en el navegador predeterminado. Las tres últimas versiones principales de los siguientes navegadores son compatibles con la IU:

   - Google Chrome
   - Mozilla Firefox
   - Microsoft Edge
   - Apple Safari para macOS

   > **Note**
   >
   > Para una mejor experiencia, recomendamos Google Chrome o Mozilla Firefox para acceder a la IU de IDT para FreeRTOS. La IU no admite Microsoft Internet Explorer.

> **Important**
>
> Debe configurar sus credenciales de AWS antes de abrir la IU. Si no ha configurado sus credenciales, cierre la ventana del navegador de la IU de IDT para FreeRTOS, siga los pasos que se indican en Configurar credenciales de AWS y, a continuación, vuelva a abrir la IU de IDT para FreeRTOS.

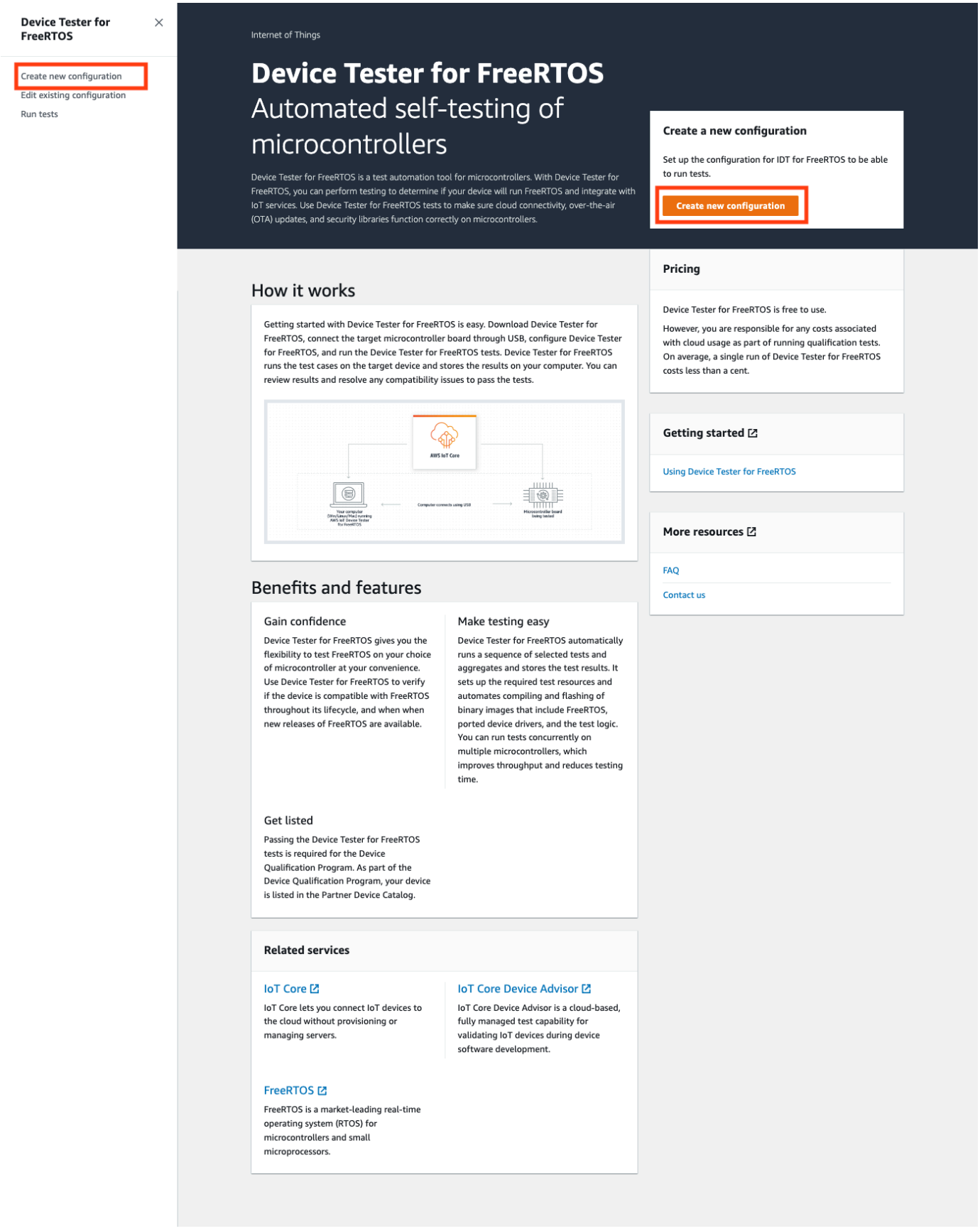
Creación de una nueva configuración

Si es la primera vez que lo usa, debe crear una nueva configuración para configurar los archivos de configuración JSON que IDT para FreeRTOS necesita para ejecutar las pruebas. A continuación, puede ejecutar pruebas o modificar la configuración creada.

Para ver ejemplos de los archivos `config.json`, `device.json` y `userdata.json`, consulte Preparación para probar su placa de microcontrolador por primera vez.

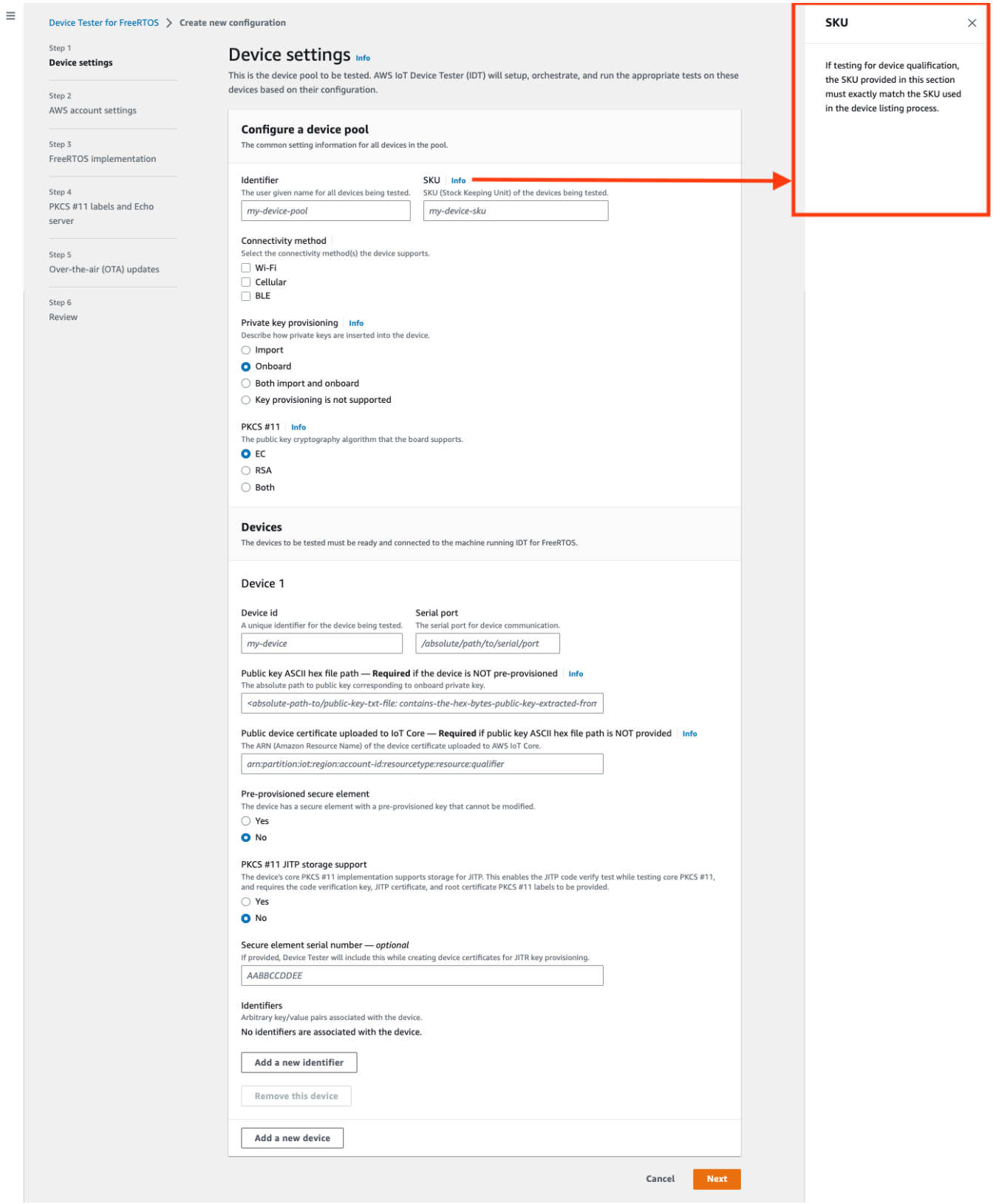
Para crear una nueva configuración

1. En la IU de IDT para FreeRTOS, abra el menú de navegación y elija Crear nueva configuración.

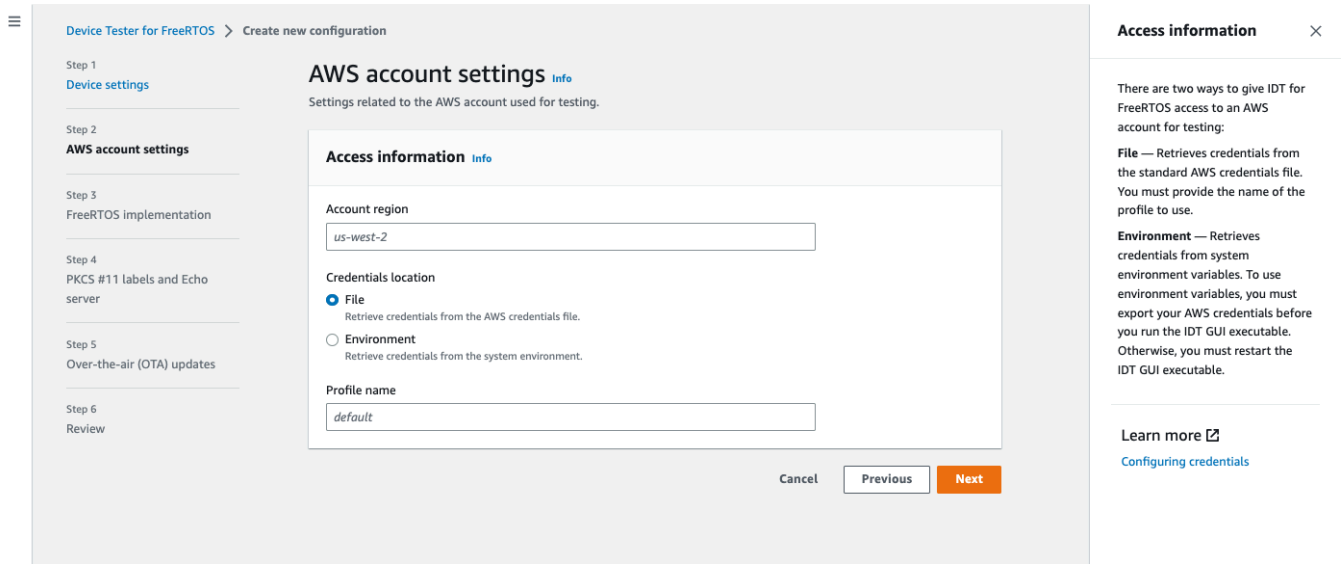

Device Tester for FreeRTOS
Create new configuration
Edit existing configuration
Run tests
Internet of Things
Device Tester for FreeRTOS
Automated self-testing of microcontrollers
Device Tester for FreeRTOS is a test automation tool for microcontrollers. With Device Tester for FreeRTOS, you can perform testing to determine if your device will run FreeRTOS and integrate with IoT services. Use Device Tester for FreeRTOS tests to make sure cloud connectivity, over-the-air (OTA) updates, and security libraries function correctly on microcontrollers.
Create a new configuration
Set up the configuration for IDT for FreeRTOS to be able to run tests.
Create new configuration
How it works
Getting started with Device Tester for FreeRTOS is easy. Download Device Tester for FreeRTOS, connect the target microcontroller board through USB, configure Device Tester for FreeRTOS, and run the Device Tester for FreeRTOS tests. Device Tester for FreeRTOS runs the test cases on the target device and stores the results on your computer. You can review results and resolve any compatibility issues to pass the tests.
AWS IoT Core
Computer connects using USB
Benefits and features
Gain confidence
Device Tester for FreeRTOS gives you the flexibility to test FreeRTOS on your choice of microcontroller at your convenience. Use Device Tester for FreeRTOS to verify if the device is compatible with FreeRTOS throughout its lifecycle, and when when new releases of FreeRTOS are available.
Make testing easy
Device Tester for FreeRTOS automatically runs a sequence of selected tests and aggregates and stores the test results. It sets up the required test resources and automates compiling and flashing of binary images that include FreeRTOS, ported device drivers, and the test logic. You can run tests concurrently on multiple microcontrollers, which improves throughput and reduces testing time.
Get listed
Passing the Device Tester for FreeRTOS tests is required for the Device Qualification Program. As part of the Device Qualification Program, your device is listed in the Partner Device Catalog.
Related services
IoT Core
IoT Core lets you connect IoT devices to the cloud without provisioning or managing servers.
IoT Core Device Advisor
IoT Core Device Advisor is a cloud-based, fully managed test capability for validating IoT devices during device software development.
FreeRTOS
FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors.
Pricing
Device Tester for FreeRTOS is free to use.
However, you are responsible for any costs associated with cloud usage as part of running qualification tests. On average, a single run of Device Tester for FreeRTOS costs less than a cent.
Getting started
Using Device Tester for FreeRTOS
More resources
FAQ
Contact us

2. Siga las instrucciones del asistente de configuración para introducir los ajustes de configuración de IDT que se utilizan para ejecutar las pruebas de calificación. El asistente configura los siguientes ajustes en los archivos de configuración JSON ubicados en el directorio *devicetester-extract-location*/config.

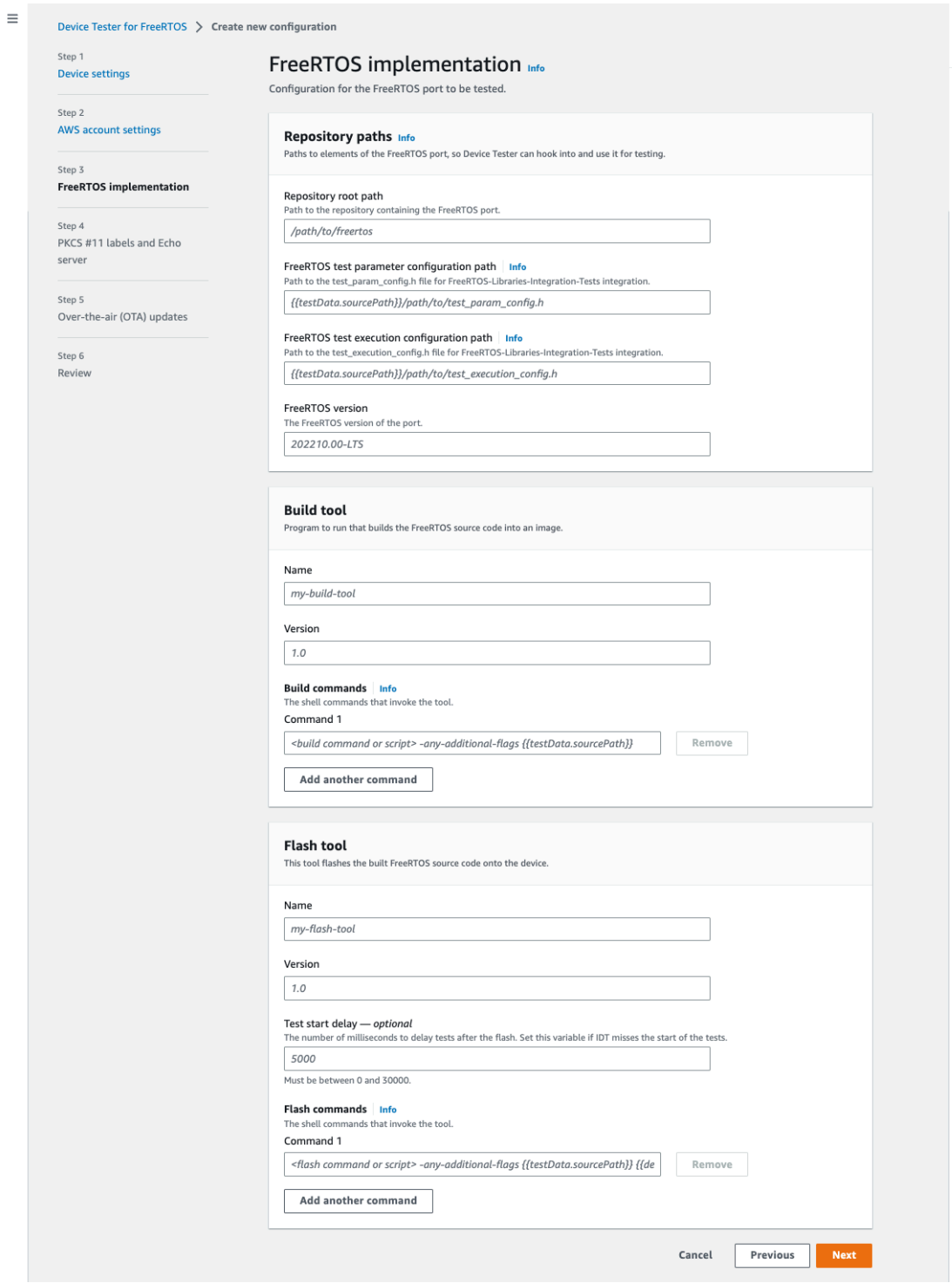
   - Configuración de dispositivos - La configuración del grupo de dispositivos para los dispositivos que se van a probar. Estos ajustes se configuran en los campos `id` y `sku`, y el bloque de dispositivos para el grupo de dispositivos en el archivo `config.json`.


Device Tester for FreeRTOS > Create new configuration
Step 1
Device settings
Step 2
AWS account settings
Step 3
FreeRTOS implementation
Step 4
PKCS #11 labels and Echo server
Step 5
Over-the-air (OTA) updates
Step 6
Review
Device settings Info
This is the device pool to be tested. AWS IoT Device Tester (IDT) will setup, orchestrate, and run the appropriate tests on these devices based on their configuration.
Configure a device pool
The common setting information for all devices in the pool.
Identifier
The user given name for all devices being tested.
my-device-pool
SKU Info
SKU (Stock Keeping Unit) of the devices being tested.
my-device-sku
Connectivity method
Select the connectivity method(s) the device supports.
Wi-Fi
Cellular
BLE
Private key provisioning Info
Describe how private keys are inserted into the device.
Import
Onboard
Both import and onboard
Key provisioning is not supported
PKCS #11 Info
The public key cryptography algorithm that the board supports.
EC
RSA
Both
Devices
The devices to be tested must be ready and connected to the machine running IDT for FreeRTOS.
Device 1
Device id
A unique identifier for the device being tested.
my-device
Serial port
The serial port for device communication.
/absolute/path/to/serial/port
Public key ASCII hex file path — Required if the device is NOT pre-provisioned Info
The absolute path to public key corresponding to onboard private key.
<absolute-path-to/public-key-txt-file: contains-the-hex-bytes-public-key-extracted-from
Public device certificate uploaded to IoT Core — Required if public key ASCII hex file path is NOT provided Info
The ARN (Amazon Resource Name) of the device certificate uploaded to AWS IoT Core.
arn:partition:iot:region:account-id:resourcetype:resource:qualifier
Pre-provisioned secure element
The device has a secure element with a pre-provisioned key that cannot be modified.
Yes
No
PKCS #11 JITP storage support
The device's core PKCS #11 implementation supports storage for JITP. This enables the JITP code verify test while testing core PKCS #11, and requires the code verification key, JITP certificate, and root certificate PKCS #11 labels to be provided.
Yes
No
Secure element serial number — optional
If provided, Device Tester will include this while creating device certificates for JITR key provisioning.
AABBCCDDEE
Identifiers
Arbitrary key/value pairs associated with the device.
No identifiers are associated with the device.
Add a new identifier
Remove this device
Add a new device
Cancel
Next
SKU
If testing for device qualification, the SKU provided in this section must exactly match the SKU used in the device listing process.

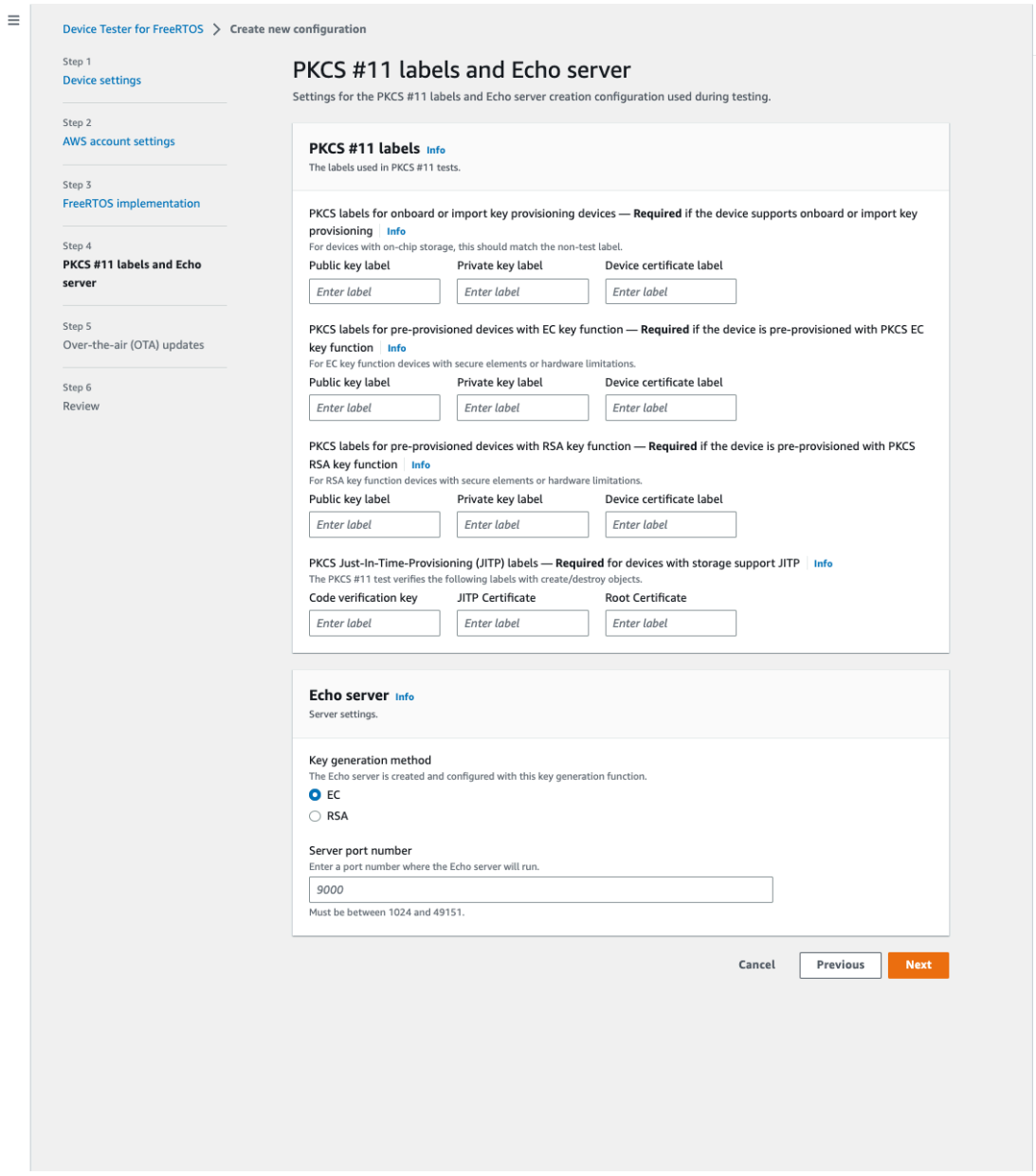
- Configuración de la cuenta de AWS - La información de Cuenta de AWS que IDT para FreeRTOS utiliza para crear recursos de AWS durante las pruebas. Estos ajustes se configuran en el archivo `config.json`.



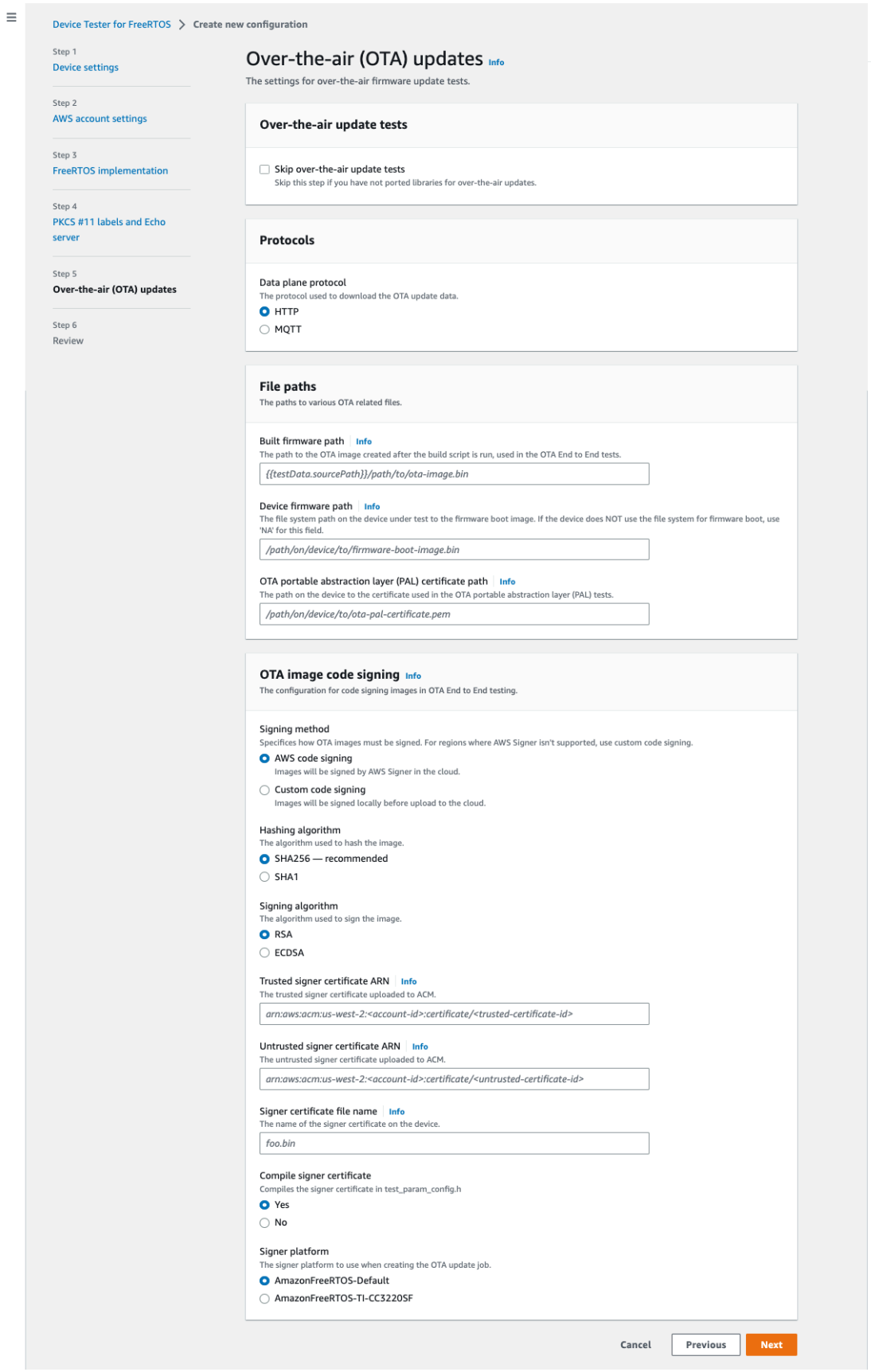
- Implementación de FreeRTOS - La ruta absoluta al repositorio y al código portado de FreeRTOS, y a la versión de FreeRTOS en la que desea ejecutar IDT FRQ. Las rutas a los archivos de encabezado de ejecución y configuración de parámetros del repositorio de GitHub de `FreeRTOS-Libraries-Integration-Tests`. Los comandos build y flash de su hardware que permiten a IDT crear e instalar las pruebas en su placa automáticamente. Estos ajustes se configuran en el archivo `userdata.json`.



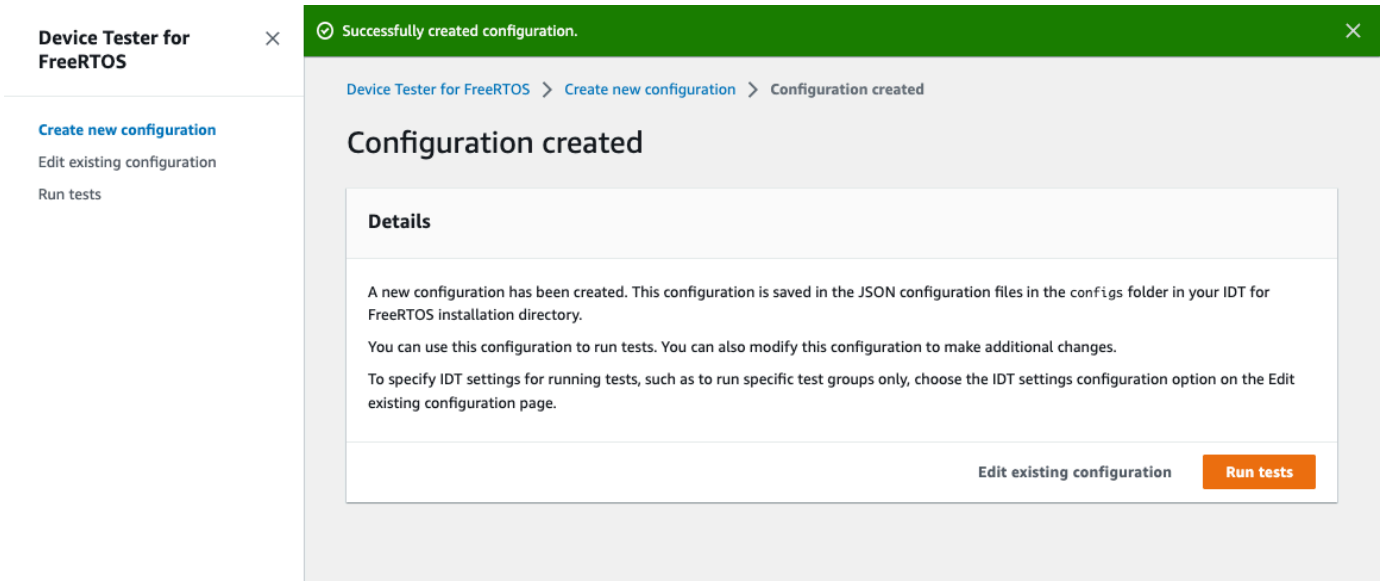
- Etiquetas PKCS 11 y servidor Echo - Las etiquetas PKCS 11 que corresponden a las claves aprovisionadas en su hardware según la funcionalidad de las clave y el método de aprovisionamiento de claves. Los ajustes de configuración del servidor echo para las pruebas de la interfaz de transporte. Estos ajustes se configuran en los archivos `userdata.json` y `device.json`.



- Actualizaciones inalámbricas (OTA) - Los ajustes que controlan las pruebas de funcionalidad de OTA. Estos ajustes se configuran en el bloque de `features` de los archivos `device.json` y `userdata.json`.


Device Tester for FreeRTOS > Create new configuration
Step 1
Device settings
Step 2
AWS account settings
Step 3
FreeRTOS implementation
Step 4
PKCS #11 labels and Echo server
Step 5
Over-the-air (OTA) updates
Step 6
Review
Over-the-air (OTA) updates Info
The settings for over-the-air firmware update tests.
Over-the-air update tests
Skip over-the-air update tests
Skip this step if you have not ported libraries for over-the-air updates.
Protocols
Data plane protocol
The protocol used to download the OTA update data.
HTTP
MQTT
File paths
The paths to various OTA related files.
Built firmware path Info
The path to the OTA image created after the build script is run, used in the OTA End to End tests.
{{testData.sourcePath}}/path/to/ota-image.bin
Device firmware path Info
The file system path on the device under test to the firmware boot image. If the device does NOT use the file system for firmware boot, use 'NA' for this field.
/path/on/device/to/firmware-boot-image.bin
OTA portable abstraction layer (PAL) certificate path Info
The path on the device to the certificate used in the OTA portable abstraction layer (PAL) tests.
/path/on/device/to/ota-pal-certificate.pem
OTA image code signing Info
The configuration for code signing images in OTA End to End testing.
Signing method
Specifices how OTA images must be signed. For regions where AWS Signer isn't supported, use custom code signing.
AWS code signing
Images will be signed by AWS Signer in the cloud.
Custom code signing
Images will be signed locally before upload to the cloud.
Hashing algorithm
The algorithm used to hash the image.
SHA256 — recommended
SHA1
Signing algorithm
The algorithm used to sign the image.
RSA
ECDSA
Trusted signer certificate ARN Info
The trusted signer certificate uploaded to ACM.
arn:aws:acm:us-west-2:<account-id>:certificate/<trusted-certificate-id>
Untrusted signer certificate ARN Info
The untrusted signer certificate uploaded to ACM.
arn:aws:acm:us-west-2:<account-id>:certificate/<untrusted-certificate-id>
Signer certificate file name Info
The name of the signer certificate on the device.
foo.bin
Compile signer certificate
Compiles the signer certificate in test_param_config.h
Yes
No
Signer platform
The signer platform to use when creating the OTA update job.
AmazonFreeRTOS-Default
AmazonFreeRTOS-TI-CC3220SF
Cancel
Previous
Next
Over-the-air (OTA) updates
IDT for FreeRTOS runs tests to verify OTA update behavior, including end-to-end (E2E) and portable abstraction layer (PAL) tests.
These tests are required to qualify a device.
Learn more
FreeRTOS OTA Update tests

3. En la página Revisar, compruebe la información de configuración.



Cuando termine de revisar la configuración, para ejecutar las pruebas de calificación, elija Ejecutar pruebas.

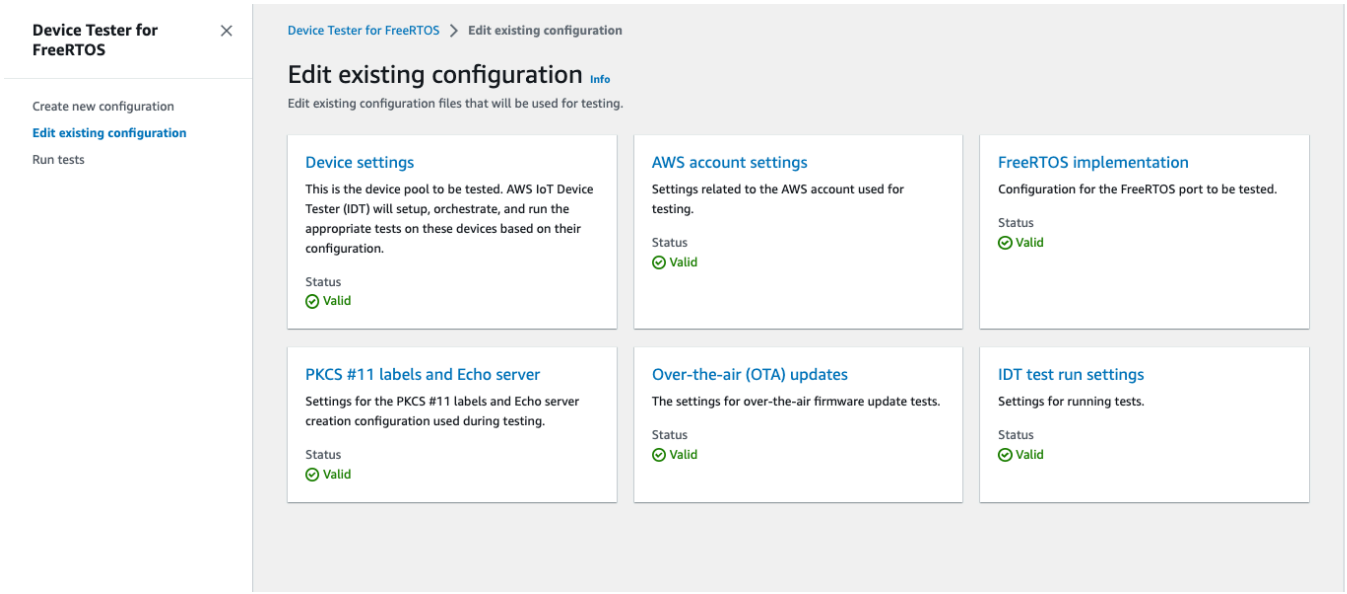
Modificación de una configuración existente

Si ya ha configurado los archivos de configuración de IDT para FreeRTOS, puede utilizar la IU de IDT para FreeRTOS para modificar la configuración existente. Los archivos de configuración existentes deben estar ubicados en el directorio *`devicetester-extract-location`*`/config`.

Para modificar una configuración

1. En la IU de IDT para FreeRTOS, abra el menú de navegación y elija Editar configuración existente.

   El panel de configuración muestra información sobre los ajustes de configuración existentes. Si una configuración es incorrecta o no está disponible, el estado de esa configuración es `Error validating configuration`.

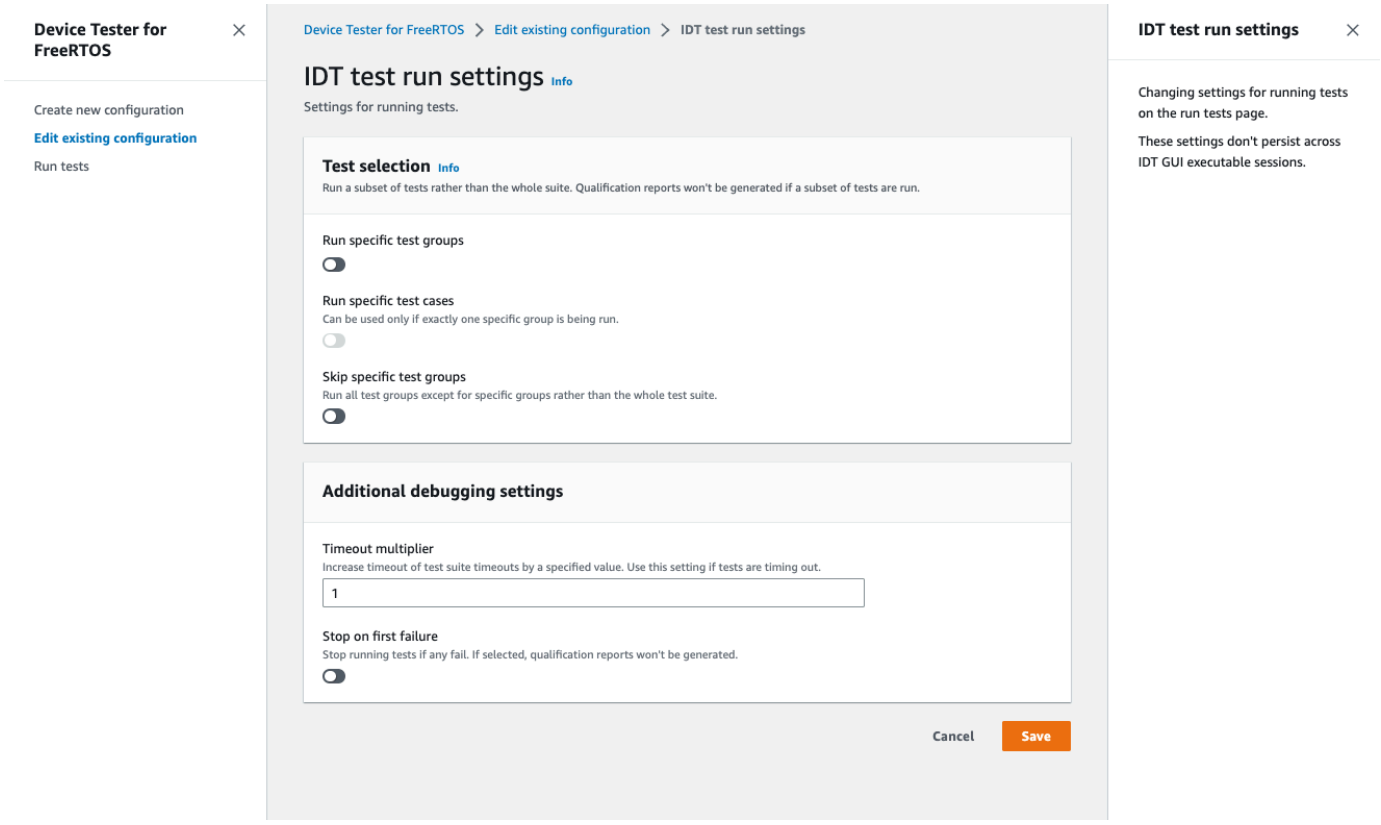


2. Complete los siguientes pasos para modificar un ajuste de configuración existente:

   a. Seleccione el nombre de un ajuste de configuración para abrir su página de ajustes.

   b. Modifique los ajustes y, a continuación, seleccione Guardar para volver a generar el archivo de configuración correspondiente.

3. Para modificar la configuración de ejecución de pruebas de IDT para FreeRTOS, elija Configuración de ejecución de pruebas de IDT en la vista de edición:



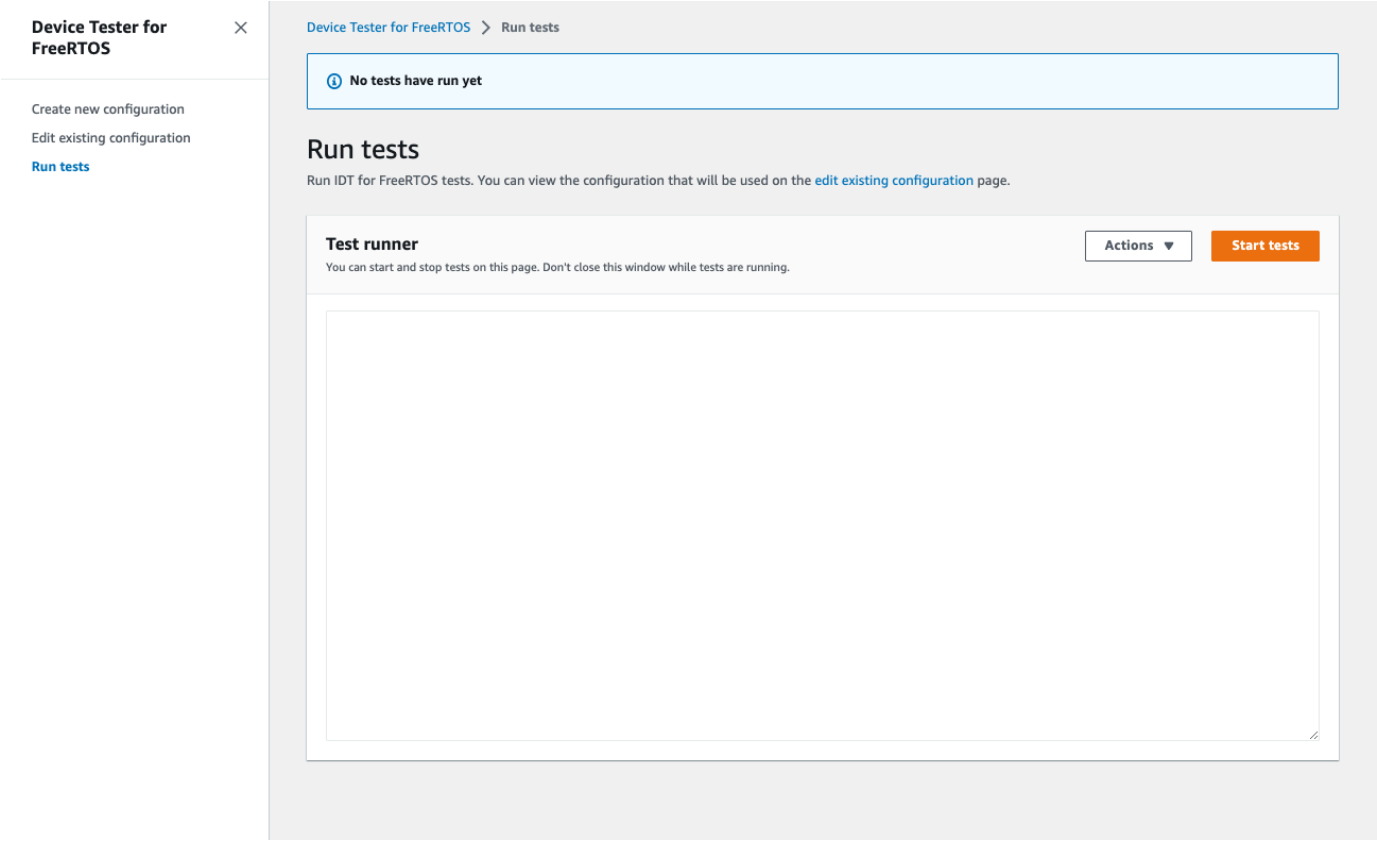
Cuando termine de modificar la configuración, compruebe que todas las opciones de configuración pasen la validación. Si el estado de cada parámetro de configuración es `Valid`, puede ejecutar las pruebas de calificación con esta configuración.

Ejecución de pruebas de calificación

Después de crear una configuración para la IU de IDT para FreeRTOS, puede ejecutar las pruebas de calificación.

Para ejecutar pruebas de calificación

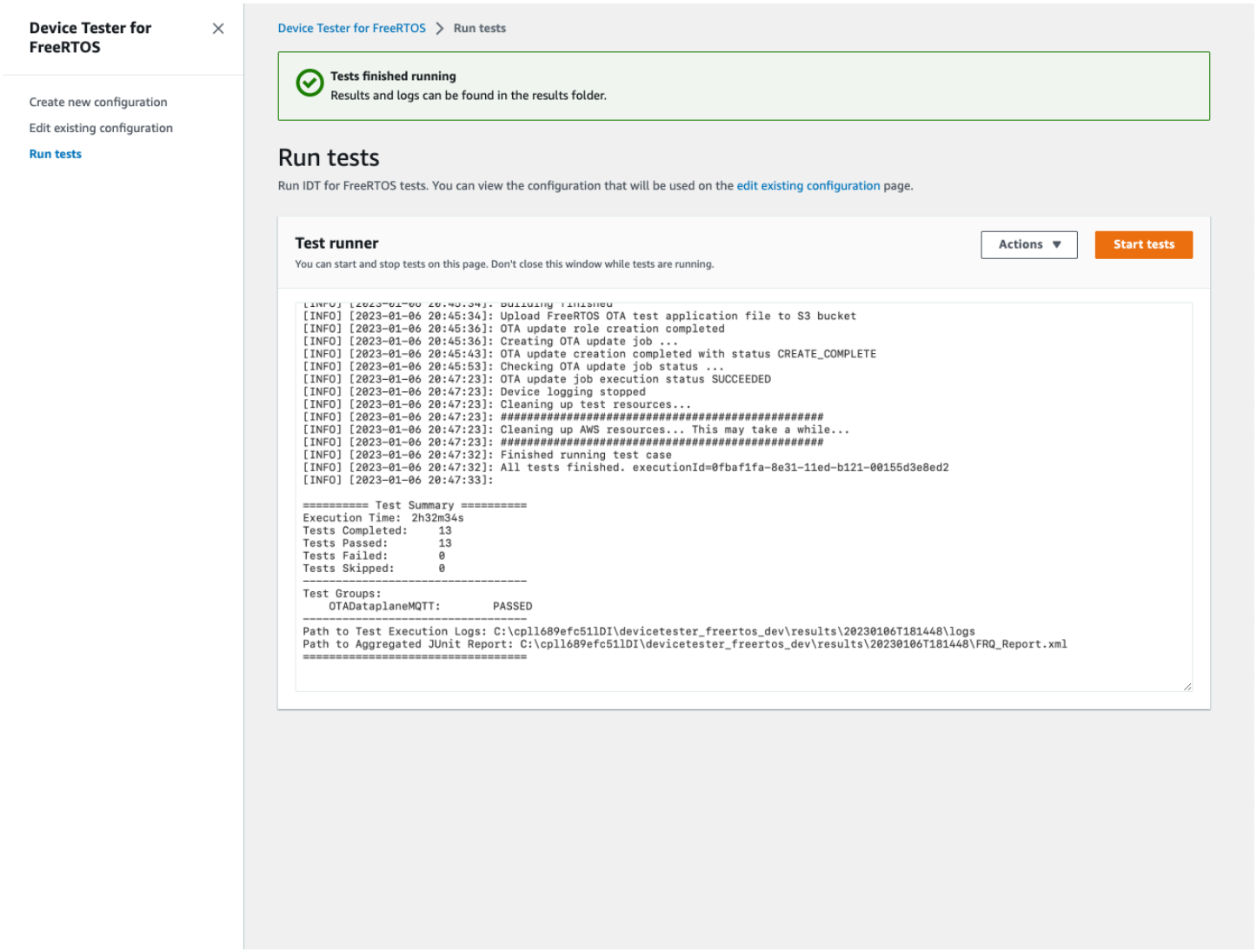
1. En el menú de navegación, elija Ejecutar pruebas.
2. Seleccione Iniciar pruebas para iniciar la ejecución de la prueba. De forma predeterminada, todas las pruebas aplicables se ejecutan para la configuración del dispositivo. IDT para FreeRTOS genera un informe de calificación cuando finalizan todas las pruebas.



IDT para FreeRTOS ejecuta las pruebas de calificación. A continuación, muestra el resumen de la ejecución de la prueba y cualquier error en la consola de Ejecutor de prueba. Una vez finalizada la ejecución de la prueba, puede ver los resultados y los registros de la prueba desde las siguientes ubicaciones:

- Los resultados de las pruebas se encuentran en el directorio *devicetester-extract-location*`/results/`*execution-id*.
- Los registros de las pruebas se encuentran en el directorio *devicetester-extract-location*`/results/`*execution-id*`/logs`.

Para obtener más información sobre los resultados de las pruebas, consulte Descripción de los resultados y de los registros.



Ejecución del conjunto de calificación de FreeRTOS 2.0

Utilice el ejecutable de AWS IoT Device Tester para FreeRTOS para interactuar con IDT para FreeRTOS. Los ejemplos de línea de comandos siguientes le muestran como ejecutar las pruebas de cualificación para un grupo de dispositivos (un conjunto de dispositivos idénticos).

IDT v4.5.2 and later

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --group-id group-id  \
    --pool-id your-device-pool \
    --test-id test-id  \
    --userdata userdata.json
```

Ejecuta un conjunto de pruebas en un grupo de dispositivos. El archivo `userdata.json` debe estar ubicado en el directorio *devicetester_extract_location*/`devicetester_freertos_`*[win|mac|linux]*`/configs/`.

> **Note**
>
> Si ejecuta IDT para FreeRTOS en Windows, utilice barras diagonales (/) para especificar la ruta al archivo `userdata.json`.

Utilice el siguiente comando para ejecutar un grupo de prueba específico:

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1.99.0  \
    --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

Los parámetros `suite-id` y `pool-id` son opcionales si está ejecutando un conjunto de pruebas único en un único grupo de dispositivos (es decir, tiene un único grupo de dispositivos definido en el archivo `device.json`).

Utilice el siguiente comando para ejecutar un caso de prueba específico:

```
devicetester_[linux | mac | win_x86-64] run-suite  \
    --group-id group-id  \
    --test-id test-id
```

Puede utilizar el comando `list-test-cases` para ver los casos de prueba en un grupo de pruebas.

Opciones de la línea de comandos de IDT para FreeRTOS

group-id

(Opcional) 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.

pool-id

(Opcional) El grupo de dispositivos que se va a probar. Es necesario si define varios grupos de dispositivos en `device.json`. Si solo tiene un grupo de dispositivos, puede omitir esta opción.

suite-id

(Opcional) La versión del conjunto de pruebas que se va a ejecutar. Si no se especifica, IDT utiliza la versión más reciente del directorio de pruebas del sistema.

test-id

(Opcional) Las pruebas que se van a ejecutar, como una lista separada por comas. Si se especifica, `group-id` debe especificar un solo grupo.

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id FreeRTOSVersion --
test-id FreeRTOSVersion
```

h

Utilice la opción de ayuda para obtener más información sobre las opciones de `run-suite`.

Example

Ejemplo

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

Comandos de IDT para FreeRTOS

El comando de IDT para FreeRTOS admite las siguientes operaciones:

IDT v4.5.2 and later

`help`

Enumera información acerca del comando especificado.

`list-groups`

Muestra los grupos de un conjunto determinado.

`list-suites`

Muestra los conjuntos disponibles.

`list-supported-products`

Muestra los productos compatibles y las versiones del conjunto de pruebas.

`list-supported-versions`

Muestra las versiones de FreeRTOS y del conjunto de pruebas compatibles con la versión actual de IDT.

`list-test-cases`

Muestra los casos de prueba de un grupo especificado.

`run-suite`

Ejecuta un conjunto de pruebas en un grupo de dispositivos.

Utilice la opción `--suite-id` para especificar una versión del conjunto de pruebas u omítala para utilizar la versión más reciente en el sistema.

Utilice el `--test-id` para ejecutar un caso de prueba individual.

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id FreeRTOSVersion --
test-id FreeRTOSVersion
```

> **Note**
>
> A partir de IDT v3.0.0, IDT comprueba en línea los conjuntos de pruebas más recientes. Para obtener más información, consulte Versiones del conjunto de pruebas.

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, escribe un resumen de ejecución

de la prueba en la consola y genera dos informes de prueba. Estos informes se pueden encontrar en *devicetester-extract-location*/results/*execution-id*/. Ambos informes capturan los resultados de la ejecución del conjunto de pruebas de cualificación.

awsiotdevicetester_report.xml es el informe de prueba de calificación que envía a AWS para mostrar su dispositivo en el Catálogo de dispositivos de socios de AWS. El informe contiene los componentes siguientes:

- La versión IDT para FreeRTOS.
- La versión de FreeRTOS que se ha probado.
- Las características de FreeRTOS que admite el dispositivo en función de las pruebas superadas.
- El SKU y el nombre de dispositivo especificado en el archivo device.json.
- Las características del dispositivo especificado en el archivo device.json.
- El resumen de agregación de los resultados de casos de prueba.
- Un desglose de los resultados de los casos de prueba por bibliotecas que se probaron en función de las características de los dispositivos.

El FRQ_Report.xml es un informe en formato JUnit XML estándar. Puede integrarlo en las plataformas CI y CD, como Jenkins, Bamboo, etc. El informe contiene los componentes siguientes:

- Un resumen de agregación de los resultados de casos de prueba.
- Un desglose de los resultados de los casos de prueba por bibliotecas que se probaron en función de las características de los dispositivos.

Interpretación de los resultados de IDT para FreeRTOS

La sección del informe en awsiotdevicetester_report.xml o FRQ_Report.xml muestra los resultados de las pruebas que se ejecutan.

La primera etiqueta XML <testsuites> contiene el resumen general de la ejecución de las pruebas. Por ejemplo:

```
<testsuites name="FRQ results" time="5633" tests="184" 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 casos de prueba ejecutados.

`failures`

El número de casos de prueba que se ejecutaron, pero que no se superaron.

`errors`

El número de casos de prueba que IDT para FreeRTOS no ha podido ejecutar.

`disabled`

Este atributo no se utiliza y se puede omitir.

Si no hay errores de caso de prueba, el dispositivo cumple los requisitos técnicos para ejecutar FreeRTOS y puede interoperar con servicios de AWS IoT. Si decide mostrar su dispositivo en el Catálogo de dispositivos de socios de AWS, puede utilizar este informe como prueba de calificación.

Si se producen errores en el caso de prueba, puede identificar el caso de prueba fallido revisando las etiquetas XML `<testsuites>`. Las etiquetas XML `<testsuite>` dentro de la etiqueta `<testsuites>` muestran el resumen del resultado de caso de prueba de un grupo de prueba.

```
<testsuite name="FreeRTOSVersion" package="" tests="1" failures="0"
time="2" disabled="0" errors="0" skipped="0">
```

El formato es similar a la etiqueta `<testsuites>`, pero con un atributo denominado `skipped` que no se utiliza y que se puede pasar por alto. Dentro de cada etiqueta XML `<testsuite>`, hay etiquetas `<testcase>` para cada uno de los casos de prueba ejecutados para un grupo de prueba. Por ejemplo:

```
<testcase classname="FRQ FreeRTOSVersion" name="FreeRTOSVersion"
attempts="1"></testcase>
```

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="core-freertos" value="not-supported" type="required"></feature>
```

Las características marcadas como `optional` no son necesarias para la cualificación. Los siguientes fragmentos muestran características opcionales:

```
<feature name="ota-dataplane-mqtt" value="not-supported" type="optional"></feature>
<feature name="ota-dataplane-http" value="not-supported" type="optional"></feature>
```

Si no hay errores de pruebas para las características requeridas, el dispositivo cumple los requisitos técnicos para ejecutar FreeRTOS y puede interoperar con servicios de AWS IoT. Si quiere mostrar su dispositivo en el Catálogo de dispositivos de socios de AWS, puede utilizar este informe como prueba de calificación.

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="FreeRTOSVersion" package="" tests="1" failures="1" time="2"
 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="FreeRTOSVersion" name="FreeRTOSVersion"></testcase>
```

Atributos que se utilizan en la etiqueta **`<testcase>`**

`name`

El nombre del caso de prueba.

`attempts`

Las veces que IDT para FreeRTOS ha ejecutado la 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="FRQ FreeRTOSVersion" name="FreeRTOSVersion">
    <failure type="Failure">Reason for the test case failure</failure>
    <error>Reason for the test case execution error</error>
</testcase>
```

Para obtener más información, consulte Solución de problemas.

Visualización de registros de

Encontrará los registros que IDT para FreeRTOS genera a partir de la ejecución de la prueba en *devicetester-extract-location*/results/*execution-id*/logs. Se generan dos conjuntos de registros:

- `test_manager.log`

  Contiene los registros generados a partir de IDT para FreeRTOS (por ejemplo, configuración relacionada con los registros y generación de informes).

- *test_group_id*/*test_case_id*/*test_case_id*.log

  El archivo de registro de un caso de prueba, incluida la salida del dispositivo que se está probando. El nombre que se asigna al archivo de registro depende del grupo de prueba y del caso de prueba ejecutado.

Uso de IDT con el conjunto de calificación FreeRTOS 1.0 (FRQ 1.0)

> **Important**
>
> A partir de octubre de 2022, AWS IoT Device Tester para la calificación de AWS IoT para FreeRTOS (FRQ) 1.0 no genera informes de calificación firmados. No puede calificar nuevos dispositivos de FreeRTOS de AWS IoT para incluirlos en el Catálogo de dispositivos de socios de AWS a través del Programa de calificación de dispositivos de AWS con las versiones IDT FRQ 1.0. Si bien no puede calificar los dispositivos FreeRTOS con IDT FRQ 1.0, puede seguir probando los dispositivos FreeRTOS con FRQ 1.0. Le recomendamos que utilice IDT FRQ 2.0 para calificar y enumerar los dispositivos FreeRTOS en el Catálogo de dispositivos de socios de AWS.

Puede utilizar la calificación de IDT para FreeRTOS para verificar que el sistema operativo de FreeRTOS funciona localmente en su dispositivo y puede comunicarse con la nube de AWS IoT. En concreto, verifica que las interfaces de la capa de portabilidad de las bibliotecas de FreeRTOS se implementan correctamente. También realiza end-to-end pruebas conAWS IoT Core. Por ejemplo, verifica si la placa es capaz de enviar o recibir mensajes MQTT y procesarlos correctamente. Las pruebas ejecutadas por IDT para FreeRTOS se definen en el repositorio FreeRTOS. GitHub

Las pruebas se ejecutan como aplicaciones integradas que se instalan en la placa. Las imágenes binarias de la aplicación incluyen FreeRTOS, las interfaces de FreeRTOS transferidas del proveedor de semiconductores y los controladores de dispositivos de la placa. El objetivo de las pruebas es verificar el correcto funcionamiento de las interfaces de FreeRTOS transferidas sobre los controladores de dispositivos.

IDT para FreeRTOS genera informes de pruebas que puede enviar a AWS IoT para añadir su hardware al Catálogo de dispositivos de socios de AWS. Para obtener más información, consulte el Programa de Calificación de Dispositivos de AWS.

IDT para FreeRTOS se ejecuta en un equipo host (Windows, macOS o Linux) que esté conectado al dispositivo que se tiene que probar. IDT ejecuta casos de prueba y agrega los resultados. También proporciona una interfaz de línea de comandos para administrar la ejecución de las pruebas.

Además de los dispositivos de pruebas, IDT para FreeRTOS crea recursos (por ejemplo, elementos de AWS IoT, grupos de FreeRTOS, funciones de Lambda, etc.) para facilitar el proceso de

cualificación. Para crear estos recursos, IDT para FreeRTOS utiliza las credenciales de AWS configuradas en `config.json` para realizar llamadas a la API en su nombre. Estos recursos se aprovisionarán en distintos momentos durante una prueba.

Cuando se ejecuta IDT para FreeRTOS en su equipo host, realiza los siguientes pasos:

1. Carga y valida su dispositivo y la configuración 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 la placa supera las pruebas necesarias para la cualificación.

Temas

Requisitos previos

En esta sección se describen los requisitos previos para probar los microcontroladores con AWS IoT Device Tester.

Descarga de FreeRTOS

Puede descargar una versión de FreeRTOS desde GitHubcon el siguiente comando:

```
git clone --branch <FREERTOS_RELEASE_VERSION> --recurse-submodules https://github.com/
aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

donde <FREERTOS_RELEASE_VERSION> es una versión de FreeRTOS (por ejemplo, 202007.00) correspondiente a una versión de IDT incluida en Versiones compatibles de AWS IoT Device Tester para FreeRTOS. Esto garantiza que dispone del código fuente completo, incluidos los submódulos, y que utiliza la versión correcta de IDT para su versión de FreeRTOS, y viceversa.

Windows tiene una limitación de longitud de ruta de 260 caracteres. La estructura de la ruta de FreeRTOS tiene muchos niveles de profundidad por lo que, si utiliza Windows, debe mantener las rutas de los archivos dentro de la limitación de 260 caracteres. Por ejemplo, clone FreeRTOS a `C:\FreeRTOS` en lugar de a `C:\Users\username\programs\projects\myproj\FreeRTOS\`.

Consideraciones para la calificación de LTS (calificación para FreeRTOS que utiliza las bibliotecas LTS)

- Para que su microcontrolador se designe como compatible con versiones basadas en soporte a largo plazo (LTS) de FreeRTOS en el Catálogo de dispositivos de socios de AWS, debe proporcionar un archivo de manifiesto. Para obtener más información, consulte la lista de verificación de calificación de FreeRTOS en la Guía de calificación de FreeRTOS.
- Para validar que su microcontrolador es compatible con las versiones basadas en LTS de FreeRTOS y poder enviarlo al Catálogo de dispositivos de socios de AWS, debe utilizar AWS IoT Device Tester (IDT) con el conjunto de pruebas FreeRTOS Qualification (FRQ) versión v1.4.x.
- La compatibilidad para las versiones basadas en LTS de FreeRTOS está limitada a la versión 202012.xx de FreeRTOS.

Descarga de IDT para FreeRTOS

Cada versión de FreeRTOS tiene una versión correspondiente de IDT para FreeRTOS para realizar pruebas de calificación. Descargue la versión apropiada de IDT para FreeRTOS en Versiones compatibles de AWS IoT Device Tester para FreeRTOS.

Extraiga IDT para FreeRTOS en una ubicación del sistema de archivos en la que tenga permisos de lectura y escritura. Dado que Microsoft Windows tiene un límite de caracteres para la longitud de la ruta de acceso, extraiga IDT para FreeRTOS en un directorio raíz, como `C:\` o `D:\`.

> **Note**
>
> No se recomienda que varios usuarios ejecuten IDT desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Esto puede provocar

bloqueos o daños en los datos. Le recomendamos que extraiga el paquete IDT a una unidad local.

Creación y configuración de una cuenta de AWS

Registro para obtener una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Cómo registrarse en 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.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación luego de completar 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.

Crear un usuario administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

Protección de su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Console como propietario de cuenta, elija Usuario raíz e ingrese el email de su Cuenta de AWS. 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 raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario administrativo

1. Activar IAM Identity Center

   Para conocer las instrucciones, consulte Habilitar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center.
2. En IAM Identity Center, otorga acceso administrativo a un usuario administrativo.

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

Cómo iniciar sesión como usuario administrativo

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

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

Política administrada de AWS IoT Device Tester

La política administrada `AWSIoTDeviceTesterForFreeRTOSFullAccess` contiene los siguientes permisos de AWS IoT Device Tester para la verificación de versiones, las características de actualización automática y la recopilación de métricas.

- `iot-device-tester:SupportedVersion`

  Otorga a AWS IoT Device Tester el permiso para obtener la lista de productos, conjuntos de pruebas y versiones de IDT compatibles.
- `iot-device-tester:LatestIdt`

Otorga a AWS IoT Device Tester el permiso para obtener la versión de IDT más reciente que se puede descargar.

- `iot-device-tester:CheckVersion`

  Otorga a AWS IoT Device Tester el permiso para comprobar la compatibilidad de las versiones de IDT, los conjuntos de pruebas y los productos.

- `iot-device-tester:DownloadTestSuite`

  Otorga a AWS IoT Device Tester el permiso para descargar conjuntos de pruebas.

- `iot-device-tester:SendMetrics`

  Otorga a AWS el permiso para recopilar métricas sobre el uso interno de AWS IoT Device Tester.

(Opcional) Instale AWS Command Line Interface

Tal vez prefiera utilizar la AWS CLI para realizar algunas operaciones. Si no tiene la AWS CLI instalada, siga las instrucciones que se indican en Instalación de la AWS CLI.

Configure la AWS CLI para la región de AWS que desea utilizar ejecutando aws configure desde una línea de comandos. Para obtener más información sobre las regiones de AWS que admiten IDT para FreeRTOS, consulte Regiones de AWS y puntos de conexión. Para obtener más información sobre aws configure, consulte Configuración rápida con aws configure.

Preparación para probar su placa de microcontrolador por primera vez

Puede utilizar IDT para FreeRTOS para realizar pruebas a medida que traslada las interfaces de FreeRTOS. Una vez que haya realizado la portabilidad las interfaces de FreeRTOS para los controladores de dispositivos de su placa, utilice AWS IoT Device Tester para ejecutar las pruebas de calificación en su placa del microcontrolador.

Añadir capas de portabilidad de bibliotecas

Para realizar la portabilidad de FreeRTOS para su dispositivo, siga las instrucciones de la Guía de portabilidad de FreeRTOS.

Configuración de sus credenciales de AWS

Debe configurar las credenciales de AWS para AWS IoT Device Tester para comunicarse con la nube de AWS. Para obtener más información, consulte Configuración de credenciales y regiones

de AWS para desarrollo. Las credenciales de AWS válidas deben especificarse en el archivo de configuración *devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/config.json.

Creación de un grupo de dispositivos en IDT para FreeRTOS

Los dispositivos que se vayan a probar se organizan en grupos de dispositivos. Cada grupo de dispositivos se compone de uno o varios dispositivos idénticos. Puede configurar IDT para FreeRTOS para probar un solo dispositivo de un grupo o varios dispositivos de un grupo. Para acelerar el proceso de calificación, IDT para FreeRTOS puede probar en paralelo dispositivos con la misma especificación. Utiliza un método de turnos rotativos para ejecutar un grupo de pruebas diferentes en todos los dispositivos de un grupo de dispositivos.

Puede añadir uno o varios dispositivos a un grupo de dispositivos editando la sección `devices` de la plantilla `device.json` en la carpeta `configs`.

> **Note**
>
> Todos los dispositivos del mismo grupo debe tener la misma especificación técnica y SKU.

Para habilitar las compilaciones en paralelo del código fuente para diferentes grupos de prueba, IDT para FreeRTOS copia el código fuente en una carpeta de resultados dentro de la carpeta extraída de IDT para FreeRTOS. Se debe hacer referencia a la ruta del código fuente en el comando build o flash con la variable `testdata.sourcePath` o `sdkPath`. IDT para FreeRTOS reemplaza esta variable con una ruta temporal del código fuente copiado. Para obtener más información, consulte Variables de IDT para FreeRTOS.

A continuación se muestra un ejemplo de un archivo `device.json` utilizado para crear un grupo de dispositivos con varios dispositivos.

```
[
    {
        "id": "pool-id",
        "sku": "sku",
        "features": [
            {
                "name": "WIFI",
                "value": "Yes | No"
            },
```

```
            {
                "name": "Cellular",
                "value": "Yes | No"
            },
            {
                "name": "OTA",
                "value": "Yes | No",
                "configs": [
                    {
                        "name": "OTADataPlaneProtocol",
                        "value": "HTTP | MQTT"
                    }
                ]
            },
            {
                "name": "BLE",
                "value": "Yes | No"
            },
            {
                "name": "TCP/IP",
                "value": "On-chip | Offloaded | No"
            },
            {
                "name": "TLS",
                "value": "Yes | No"
            },
            {
                "name": "PKCS11",
                "value": "RSA | ECC | Both | No"
            },
            {
                "name": "KeyProvisioning",
                "value": "Import | Onboard | No"
            }
        ],

        "devices": [
          {
            "id": "device-id",
            "connectivity": {
              "protocol": "uart",
              "serialPort": "/dev/tty*"
            },
```

```
            **********Remove the section below if the device does not support onboard
key generation**************
            "secureElementConfig" : {
              "publicKeyAsciiHexFilePath": "absolute-path-to/public-key-txt-file:
contains-the-hex-bytes-public-key-extracted-from-onboard-private-key",
              "secureElementSerialNumber": "secure-element-serialNo-value",
              "preProvisioned"           : "Yes | No"
            },

*****************************************************************************************
            "identifiers": [
              {
                "name": "serialNo",
                "value": "serialNo-value"
              }
            ]
          }
        ]
    }
]
```

En el archivo `device.json` se utilizan los siguientes atributos:

`id`

Un ID alfanumérico definido por el usuario que identifica de manera exclusiva a un grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben ser del mismo tipo. Cuando se ejecuta un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo.

`sku`

Un valor alfanumérico que identifica de forma única la placa que está probando. El SKU se utiliza para realizar un seguimiento de placas cualificadas.

> **Note**
>
> Si desea enumerar la placa en el Catálogo de dispositivos de socios de AWS, la SKU que especifique aquí debe coincidir con la SKU que utilice en el proceso de enumeración.

`features`

Una matriz que contiene las funciones compatibles del dispositivo. AWS IoT Device Tester utiliza esta información para seleccionar las pruebas de calificación que se van a ejecutar.

Los valores admitidos son:

`TCP/IP`

Indica si la placa admite una pila TCP/IP y si es compatible en chip (MCU) o se descarga en otro módulo. TCP/IP es necesario para la cualificación.

`WIFI`

Indica si la placa tiene capacidades Wi-Fi. Se debe establecer en `No` si `Cellular` se establece en `Yes`.

`Cellular`

Indica si la placa tiene capacidad móvil. Se debe establecer en `No` si `WIFI` se establece en `Yes`. Si esta función está configuradaYes, la FullSecureSockets prueba se ejecutará con instancias EC2 de AWS t2.micro, lo que puede suponer costes adicionales para su cuenta. Para obtener más información, consulte Precios de Amazon EC2.

`TLS`

Indica si la placa admite TLS. TLS es necesario para la cualificación.

`PKCS11`

Indica el algoritmo de criptografía de clave pública que admite la placa. PKCS11 es necesario para la cualificación. Los valores admitidos son `ECC`, `RSA`, `Both` y `No`. `Both` indica que la placa admite los algoritmos `ECC` y `RSA`.

`KeyProvisioning`

Indica el método para escribir un certificado de cliente X.509 de confianza en la placa. Los valores admitidos son `Import`, `Onboard` y `No`. El aprovisionamiento de claves es necesario para la cualificación.

- Utilice `Import` si su placa permite la importación de claves privadas. IDT creará una clave privada y la compilará en el código fuente de FreeRTOS.
- Utilice `Onboard` si la placa admite la generación de claves privadas integrada (por ejemplo, si su dispositivo tiene un elemento seguro o si prefiere generar su propio par de claves de dispositivo y certificado). Procure añadir un elemento `secureElementConfig` en cada una

de las secciones del dispositivo e introduzca la ruta absoluta del archivo de claves públicas en el campo `publicKeyAsciiHexFilePath`.

- Si la placa no admite el aprovisionamiento de claves, utilice `No`.

`OTA`

Indica si su placa admite la funcionalidad de actualización over-the-air (OTA). El atributo `OtaDataPlaneProtocol` indica qué protocolo de plano de datos de OTA admite el dispositivo. El atributo se omite si el dispositivo no admite la característica OTA. Cuando `"Both"` se selecciona, el tiempo de ejecución de las pruebas OTA se prolonga debido a la ejecución de MQTT, HTTP y pruebas mixtas.

> **Note**
>
> A partir de la versión 4.1.0 de IDT, `OtaDataPlaneProtocol` solo acepta `HTTP` y `MQTT` como valores admitidos.

`BLE`

Indica si la placa admite Bluetooth de bajo consumo (BLE).

`devices.id`

Un identificador único y definido por el usuario para el dispositivo que se está probando.

`devices.connectivity.protocol`

El protocolo de comunicación que se usará para la comunicación con este dispositivo. Valor admitido: `uart`.

`devices.connectivity.serialPort`

El puerto de serie del equipo host utilizado para conectarse a los dispositivos que se van a probar.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

La ruta absoluta del archivo que contiene la clave pública de bytes hexadecimales extraída de la clave privada integrada.

Formato de ejemplo:

```
3059 3013 0607 2a86 48ce 3d02 0106 082a
```

```
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

Si la clave pública tiene el formato .der, puede codificar directamente la clave pública en formato hexadecimal para generar el archivo hexadecimal.

Ejemplo de comando para que la clave pública .der genere un archivo hexadecimal:

```
xxd -p pubkey.der > outFile
```

Si la clave pública tiene el formato .pem, puede extraer la parte codificada en base64, decodificarla en formato binario y, a continuación, codificarla en formato hexadecimal para generar el archivo hexadecimal.

Por ejemplo, utilice estos comandos para generar un archivo hexadecimal para una clave pública .pem:

1. Saque la parte de la clave codificada en base64 (elimine el encabezado y el pie de página) y guárdela en un archivo, por ejemplo, con el nombre `base64key`, y ejecute este comando para convertirla al formato .der:

   ```
   base64 –decode base64key > pubkey.der
   ```

2. Ejecute el comando `xxd` para convertirla a formato hexadecimal.

   ```
   xxd -p pubkey.der > outFile
   ```

`devices.secureElementConfig.SecureElementSerialNumber`

(Opcional) Número de serie del elemento seguro. Indique este campo cuando se imprima el número de serie junto con la clave pública del dispositivo al ejecutar el proyecto de demostración/prueba de FreeRTOS.

`devices.secureElementConfig.preProvisioned`

(Opcional) Indique el valor “Sí” si el dispositivo tiene un elemento seguro aprovisionado previamente con credenciales bloqueadas, que no puede importar, crear ni destruir objetos.

Esta configuración solo se aplica cuando `features` tiene `KeyProvisioning` establecido en "Incorporación" y `PKCS11` está establecido en "ECC".

`identifiers`

(Opcional) Una matriz de pares de nombre-valor arbitrarios. Puede utilizar estos valores en los comandos Build y Flash descritos en la siguiente sección.

Configurar ajustes de Build, Flash y Test

Para que IDT para FreeRTOS pueda compilar e instalar pruebas en su placa de forma automática, debe configurar IDT para que ejecute los comandos de compilación e instalación para su hardware. Los ajustes de compilación e instalación se configuran en el archivo de plantilla `config` que se encuentra en la carpeta `userdata.json`.

Configurar ajustes para probar dispositivos

Los ajustes de compilación, instalación y prueba se realizan en el archivo `configs/userdata.json`. Se admite la configuración del servidor Echo cargando los certificados y las claves del cliente y del servidor en `customPath`. Para obtener más información, consulte Configuración de un servidor echo en la Guía de portabilidad de FreeRTOS. El siguiente ejemplo de JSON muestra cómo puede configurar IDT para FreeRTOS para probar varios dispositivos:

```
{
    "sourcePath": "/absolute-path-to/freertos",
    "vendorPath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-name",
    // ***********The sdkConfiguration block below is needed if you are not using the
 default, unmodified FreeRTOS repo.
    // In other words, if you are using the default, unmodified FreeRTOS repo then
 remove this block***************
    "sdkConfiguration": {
        "name": "sdk-name",
        "version": "sdk-version",
        "path": "/absolute-path-to/sdk"
    },
    "buildTool": {
        "name": "your-build-tool-name",
        "version": "your-build-tool-version",
        "command": [
            "{{config.idtRootPath}}/relative-path-to/build-parallel.sh
 {{testData.sourcePath}} {{enableTests}}"
        ]
```

```
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "/{{config.idtRootPath}}/relative-path-to/flash-parallel.sh
{{testData.sourcePath}} {{device.connectivity.serialPort}} {{buildImageName}}"
        ],
        "buildImageInfo" : {
            "testsImageName": "tests-image-name",
            "demosImageName": "demos-image-name"
        }
    },
    "testStartDelayms": 0,
    "clientWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA |
eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
    },
    "testWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA |
eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
    },
    //**********
    //This section is used to start echo server based on server certificate generation
method,
    //When certificateGenerationMethod is set as Automatic specify the eccCurveFormat
to generate certifcate and key based on curve format,
    //When certificateGenerationMethod is set as Custom specify the certificatePath and
PrivateKeyPath to be used to start echo server
    //**********
    "echoServerCertificateConfiguration": {
      "certificateGenerationMethod": "Automatic | Custom",
      "customPath": {
          "clientCertificatePath":"/path/to/clientCertificate",
          "clientPrivateKeyPath": "/path/to/clientPrivateKey",
          "serverCertificatePath":"/path/to/serverCertificate",
          "serverPrivateKeyPath": "/path/to/serverPrivateKey"
      },
    "eccCurveFormat": "P224 | P256 | P384 | P521"
    },
```

```
    "echoServerConfiguration": {
        "securePortForSecureSocket": 33333, // Secure tcp port used by SecureSocket
 test. Default value is 33333. Ensure that the port configured isn't blocked by the
 firewall or your corporate network
        "insecurePortForSecureSocket": 33334, // Insecure tcp port used by SecureSocket
 test. Default value is 33334. Ensure that the port configured isn't blocked by the
 firewall or your corporate network
        "insecurePortForWiFi": 33335 // Insecure tcp port used by Wi-Fi test. Default
 value is 33335. Ensure that the port configured isn't blocked by the firewall or your
 corporate network
    },
    "otaConfiguration": {
        "otaFirmwareFilePath": "{{testData.sourcePath}}/relative-path-to/ota-image-
generated-in-build-process",
        "deviceFirmwareFileName": "ota-image-name-on-device",
        "otaDemoConfigFilePath": "{{testData.sourcePath}}/relative-path-to/ota-demo-
config-header-file",
        "codeSigningConfiguration": {
            "signingMethod": "AWS | Custom",
            "signerHashingAlgorithm": "SHA1 | SHA256",
            "signerSigningAlgorithm": "RSA | ECDSA",
            "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
            "signerCertificateFileName": "signerCertificate-file-name",
            "compileSignerCertificate": boolean,
            // ***********Use signerPlatform if you choose aws for
 signingMethod***************
            "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
            "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
            // ***********Use signCommand if you choose custom for
 signingMethod***************
            "signCommand": [
                "/absolute-path-to/sign.sh {{inputImageFilePath}}
 {{outputSignatureFilePath}}"
            ]
        }
    },
    // ***********Remove the section below if you're not configuring
 CMake***************
    "cmakeConfiguration": {
        "boardName": "board-name",
        "vendorName": "vendor-name",
        "compilerName": "compiler-name",
```

```
        "frToolchainPath": "/path/to/freertos/toolchain",
        "cmakeToolchainPath": "/path/to/cmake/toolchain"
    },
    "freertosFileConfiguration": {
        "required": [
            {
                "configName": "pkcs11Config",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/core_pkcs11_config.h"
            },
            {
                "configName": "pkcs11TestConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/iot_test_pkcs11_config.h"
            }
        ],
        "optional": [
            {
                "configName": "otaAgentTestsConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/ota_config.h"
            },
            {
                "configName": "otaAgentDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_demos/config_files/ota_config.h"
            },
            {
                "configName": "otaDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_demos/config_files/ota_demo_config.h"
            }
        ]
    }
}
```

A continuación se indican los atributos usados en `userdata.json`:

`sourcePath`

La ruta a la raíz del código fuente de FreeRTOS trasladado. Para las pruebas en paralelo con un SDK, `sourcePath` se puede establecer mediante el marcador de posición `{{userData.sdkConfiguration.path}}`. Por ejemplo:

```
{ "sourcePath":"{{userData.sdkConfiguration.path}}/freertos" }
```

vendorPath

La ruta al código de FreeRTOS específico del proveedor. Para las pruebas en serie, `vendorPath` se puede establecer como una ruta absoluta. Por ejemplo:

```
{ "vendorPath":"C:/path-to-freertos/vendors/espressif/boards/esp32" }
```

Para las pruebas en paralelo, `vendorPath` se puede establecer mediante el marcador `{{testData.sourcePath}}`. Por ejemplo:

```
{ "vendorPath":"{{testData.sourcePath}}/vendors/espressif/boards/esp32" }
```

La variable `vendorPath` solo es necesaria cuando se ejecuta sin un SDK; de lo contrario, se puede eliminar.

> **Note**
>
> Al ejecutar pruebas en paralelo sin un SDK, el marcador de posición `{{testData.sourcePath}}` se debe utilizar en los campos `vendorPath`, `buildTool` y `flashTool`. Al ejecutar la prueba con un solo dispositivo, las rutas absolutas se deben utilizar en los campos `vendorPath`, `buildTool` y `flashTool`. Cuando se ejecuta con un SDK, el marcador de posición `{{sdkPath}}` debe usarse en los comandos `sourcePath`, `buildTool` y `flashTool`.

sdkConfiguration

Si va a calificar FreeRTOS con alguna modificación en la estructura de archivos y carpetas que vaya más allá de lo necesario para la portabilidad, tendrá que configurar la información del SDK en este bloque. Si no va a calificar un FreeRTOS con un FreeRTOS trasladado dentro de un SDK, puede omitir este bloque por completo.

sdkConfiguration.name

El nombre del SDK que está utilizando con FreeRTOS. Si no utiliza un SDK, puede omitir todo el bloque de `sdkConfiguration`.

sdkConfiguration.version

La versión del SDK que está utilizando con FreeRTOS. Si no utiliza un SDK, puede omitir todo el bloque de `sdkConfiguration`.

sdkConfiguration.path

La ruta absoluta al directorio del SDK que contiene el código de FreeRTOS. Si no utiliza un SDK, puede omitir todo el bloque de `sdkConfiguration`.

buildTool

La ruta completa a su script de compilación (.bat o .sh) que contiene los comandos para compilar el código fuente. Todas las referencias a la ruta del código fuente en el comando de compilación deben reemplazarse por la variable `{{testdata.sourcePath}}` de AWS IoT Device Tester y las referencias a la ruta del SDK deben reemplazarse por `{{sdkPath}}`. Utilice el marcador de posición `{{config.idtRootPath}}` para hacer referencia a la ruta de IDT absoluta o relativa.

testStartDelayms

Especifica cuántos milisegundos esperará el ejecutor de pruebas de FreeRTOS antes de empezar a ejecutar las pruebas. Esto puede resultar útil si el dispositivo que se está probando comienza a generar información de prueba importante antes de que IDT tenga la oportunidad de conectarse y empezar a registrar datos debido a una latencia de red o de otro tipo. El valor máximo permitido es de 30 000 ms (30 segundos). Este valor se aplica únicamente a los grupos de pruebas de FreeRTOS y no a otros grupos de pruebas que no utilizan el ejecutor de pruebas de FreeRTOS, como las pruebas OTA.

flashTool

Ruta completa a su script de flash (.sh o .bat) que contiene los comandos flash para su dispositivo. Todas las referencias a la ruta del código fuente en el comando de instalación deben reemplazarse por la variable `{{testdata.sourcePath}}` de IDT para FreeRTOS y todas las referencias a la ruta del SDK deben reemplazarse por la variable `{{sdkPath}}` de IDT para FreeRTOS. Utilice el marcador de posición `{{config.idtRootPath}}` para hacer referencia a la ruta de IDT absoluta o relativa.

buildImageInfo

testsImageName

El nombre del archivo creado por el comando de compilación al compilar pruebas desde la carpeta *freertos-source*`/tests`.

`demosImageName`

El nombre del archivo creado por el comando de compilación al compilar pruebas desde la carpeta *`freertos-source`*`/demos`.

`clientWifiConfig`

La configuración de wifi del cliente. Las pruebas de biblioteca Wi-Fi requieren una placa MCU para conectarse a dos puntos de acceso. (Los dos puntos de acceso pueden ser los mismos). Este atributo configura los ajustes de Wi-Fi para el primer punto de acceso. Algunos de los casos de prueba Wi-Fi esperan que el punto de acceso tenga cierto nivel de seguridad y que no estén abiertos. Asegúrese de que ambos puntos de acceso estén en la misma subred que el equipo host que ejecuta IDT.

`wifi_ssid`

El SSID de wifi.

`wifi_password`

La contraseña de wifi.

`wifiSecurityType`

El tipo de seguridad wifi utilizada. Uno de los valores:

- `eWiFiSecurityOpen`
- `eWiFiSecurityWEP`
- `eWiFiSecurityWPA`
- `eWiFiSecurityWPA2`
- `eWiFiSecurityWPA3`

> **Note**
>
> Si la placa no es compatible con wifi, debe volver a incluir la sección `clientWifiConfig` en el archivo `device.json`, pero puede omitir los valores de estos atributos.

`testWifiConfig`

La configuración de wifi de prueba. Las pruebas de biblioteca Wi-Fi requieren una placa MCU para conectarse a dos puntos de acceso. (Los dos puntos de acceso pueden ser los mismos).

Este atributo configura los ajustes de wifi para el segundo punto de acceso. Algunos de los casos de prueba Wi-Fi esperan que el punto de acceso tenga cierto nivel de seguridad y que no estén abiertos. Asegúrese de que ambos puntos de acceso estén en la misma subred que el equipo host que ejecuta IDT.

`wifiSSID`

El SSID de wifi.

`wifiPassword`

La contraseña de wifi.

`wifiSecurityType`

El tipo de seguridad wifi utilizada. Uno de los valores:

- `eWiFiSecurityOpen`
- `eWiFiSecurityWEP`
- `eWiFiSecurityWPA`
- `eWiFiSecurityWPA2`
- `eWiFiSecurityWPA3`

> **Note**
>
> Si la placa no es compatible con wifi, debe volver a incluir la sección `testWifiConfig` en el archivo `device.json`, pero puede omitir los valores de estos atributos.

`echoServerCertificateConfiguration`

El marcador de posición configurable para la generación de certificados del servidor echo para realizar pruebas de conexiones seguras. Este campo es obligatorio.

`certificateGenerationMethod`

Especifica si el certificado del servidor se genera automáticamente o se proporciona manualmente.

`customPath`

Si `certificateGenerationMethod` es “Personalizado”, `certificatePath` y `privateKeyPath` son obligatorios.

`certificatePath`

Especifica la ruta de archivo del certificado del servidor.

`privateKeyPath`

Especifica la ruta del archivo de la clave privada.

`eccCurveFormat`

Especifica el formato de curva que admite la placa. Es necesario si `PKCS11` se ha establecido en “ecc” en `device.json`. Los valores válidos son “P224”, “P256”, “P384” o “P521”.

`echoServerConfiguration`

Los puertos configurables del servidor Echo sirven para WiFi realizar pruebas de sockets seguros. Este campo es opcional.

`securePortForSecureSocket`

El puerto que se utiliza para configurar un servidor de eco con TLS para la prueba de sockets seguros. El valor predeterminado es 33333. Asegúrese de que el puerto configurado no esté bloqueado por un firewall o por la red de la empresa.

`insecurePortForSecureSocket`

El puerto que se utiliza para configurar un servidor de eco sin TLS para la prueba de sockets seguros. El valor predeterminado que se ha utilizado en la prueba es 33334. Asegúrese de que el puerto configurado no esté bloqueado por un firewall o por la red de la empresa.

`insecurePortForWiFi`

El puerto que se utiliza para configurar el servidor echo sin TLS para la WiFi prueba. El valor predeterminado que se ha utilizado en la prueba es 33335. Asegúrese de que el puerto configurado no esté bloqueado por un firewall o por la red de la empresa.

`otaConfiguration`

La configuración de OTA. [Opcional]

`otaFirmwareFilePath`

La ruta completa a la imagen de OTA creada después de la compilación. Por ejemplo, `{{testData.sourcePath}}/`*`relative-path/to/ota/image/from/source/root`*.

deviceFirmwareFileName

La ruta de archivo completa en el dispositivo de MCU donde se descargará el firmware de OTA. Algunos dispositivos no utilizan este campo, pero en cualquier caso deberá proporcionar un valor.

otaDemoConfigFilePath

La ruta completa a `aws_demo_config.h`, que se encuentra en *`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/`. Estos archivos se incluyen en la plantilla de código de portabilidad proporcionada por FreeRTOS.

codeSigningConfiguration

La configuración de firma de código.

signingMethod

El método de firma de código. Los valores posibles son `AWS` o `Custom`.

> **Note**
>
> Para las regiones de Pekín y Ningxia, utilice `Custom`. La firma de código de `AWS` no se admite en estas regiones.

signerHashingAlgorithm

El algoritmo hash admitido en el dispositivo. Los valores posibles son `SHA1` o `SHA256`.

signerSigningAlgorithm

El algoritmo de firma admitido en el dispositivo. Los valores posibles son `RSA` o `ECDSA`.

signerCertificate

Certificado de confianza utilizado para OTA.

Para el método de firma de código de AWS, utilice el nombre de recurso de Amazon (ARN) del certificado de confianza cargado en AWS Certificate Manager.

Para el método de firma de código personalizado, utilice la ruta absoluta al archivo de certificado del firmante.

Para obtener más información acerca de cómo crear un certificado de confianza, consulte Crear un certificado de firma de código.

`signerCertificateFileName`

El nombre de la ruta del certificado de firma de código en el dispositivo. Este valor debe coincidir con el nombre de archivo que proporcionó al ejecutar el comando `aws acm import-certificate`.

Para obtener más información, consulte Crear un certificado de firma de código.

`compileSignerCertificate`

Establézcalo en `true` si el certificado de verificación de firma del firmante de código no se ha aprovisionado o instalado, por lo que debe compilarse en el proyecto. AWS IoT Device Tester obtiene el certificado de confianza y lo compila en `aws_codesigner_certifiate.h`.

`untrustedSignerCertificate`

El ARN o ruta de archivo de un segundo certificado utilizado en algunas pruebas OTA como certificado que no es de confianza. Para obtener más información sobre cómo crear un certificado, consulte Creación de un certificado de firma de código.

`signerPlatform`

El algoritmo de firma y hash que AWS Code Signer utiliza al crear el trabajo de actualización de OTA. En la actualidad, los valores posibles para este campo son `AmazonFreeRTOS-TI-CC3220SF` y `AmazonFreeRTOS-Default`.

- Elija `AmazonFreeRTOS-TI-CC3220SF` si es `SHA1` y `RSA`.
- Elija `AmazonFreeRTOS-Default` si es `SHA256` y `ECDSA`.

Si necesita `SHA256` | `RSA` o `SHA1` | `ECDSA` para su configuración, contacte con nosotros para obtener ayuda adicional.

Configure `signCommand` si eligió `Custom` para `signingMethod`.

`signCommand`

El comando utilizado para realizar la firma de código personalizado. Puede encontrar la plantilla en el directorio `/configs/script_templates`.

Se necesitan dos marcadores de posición `{{inputImageFilePath}}` y `{{outputSignatureFilePath}}` en el comando. `{{inputImageFilePath}}`

es la ruta del archivo de la imagen creada por IDT que se va a firmar. `{{outputSignatureFilePath}}` es la ruta del archivo de la firma que generará el script.

`cmakeConfiguration`

Configuración de CMake [opcional]

> **Note**
>
> Para ejecutar casos de prueba de CMake, debe proporcionar el nombre de la placa, el nombre del proveedor y la `frToolchainPath` o el `compilerName`. También puede proporcionar la `cmakeToolchainPath` si tiene una ruta personalizada a la cadena de herramientas de CMake.

`boardName`

El nombre de la placa que se prueba. El nombre de la placa debe ser el mismo que el nombre de la carpeta en *`path/to/afr/source/code`*`/vendors/`*`vendor`*`/boards/`*`board`*.

`vendorName`

El nombre del proveedor de la placa que se prueba. El nombre del proveedor debe ser el mismo que el nombre de la carpeta *`path/to/afr/source/code`*`/vendors/`*`vendor`*.

`compilerName`

El nombre del compilador.

`frToolchainPath`

La ruta completa de la cadena de herramientas del compilador.

`cmakeToolchainPath`

La ruta completa de la cadena de herramientas de CMake. Este campo es opcional

`freertosFileConfiguration`

La configuración de los archivos FreeRTOS que IDT modifica antes de ejecutar las pruebas.

`required`

En esta sección se especifican las pruebas obligatorias cuyos archivos de configuración ha movido, por ejemplo, PKCS11, TLS, etc.

`configName`

El nombre de la prueba que se está configurando.

`filePath`

La ruta absoluta a los archivos de configuración en el repositorio *`freertos`*. Utilice la variable `{{testData.sourcePath}}` para definir la ruta.

`optional`

En esta sección se especifican las pruebas opcionales cuyos archivos de configuración ha movido, por ejemplo WiFi, OTA, etc.

`configName`

El nombre de la prueba que se está configurando.

`filePath`

La ruta absoluta a los archivos de configuración en el repositorio *`freertos`*. Utilice la variable `{{testData.sourcePath}}` para definir la ruta.

> **Note**
>
> Para ejecutar casos de prueba de CMake, debe proporcionar el nombre de la placa, el nombre del proveedor y la `afrToolchainPath` o el `compilerName`. También puede proporcionar la `cmakeToolchainPath` si tiene una ruta personalizada a la cadena de herramientas de CMake.

Variables de IDT para FreeRTOS

Los comandos para compilar el código y actualizar el dispositivo pueden requerir conectividad u otra información sobre los dispositivos para funcionar correctamente. AWS IoT Device Testerpermite hacer referencia a la información del dispositivo en flash y compilar comandos utilizando JsonPath. Mediante el uso de JsonPath expresiones sencillas, puede obtener la información requerida especificada en el `device.json` archivo.

Variables de ruta

IDT para FreeRTOS define las siguientes variables de ruta que se pueden utilizar en líneas de comandos y archivos de configuración:

{{testData.sourcePath}}

Amplía la ruta del código fuente. Si utiliza esta variable, se debe utilizar tanto en los comandos flash como en los comandos build.

{{sdkPath}}

Se expande al valor que tiene en su `userData.sdkConfiguration.path` cuando se utiliza en los comandos build y flash.

{{device.connectivity.serialPort}}

Amplía al puerto serie.

{{device.identifiers[?(@.name == 'serialNo')].value[0]}}

Se amplía hasta el número de serie de su dispositivo.

{{enableTests}}

Valor entero que indica si la compilación es para pruebas (valor 1) o demostraciones (valor 0).

{{buildImageName}}

El nombre de archivo de la imagen compilada por el comando de compilación.

{{otaCodeSignerPemFile}}

Archivo PEM para el firmante del código OTA.

{{config.idtRootPath}}

Se expande hasta la ruta raíz AWS IoT Device Tester. Esta variable reemplaza la ruta absoluta de IDT cuando la utilizan los comandos build y flash.

Uso de la interfaz de usuario de IDT para FreeRTOS para ejecutar el conjunto de calificación FreeRTOS

A partir de IDT v4.3.0, AWS IoT Device Tester para FreeRTOS (IDT-FreeRTOS) incluye una interfaz de usuario basada en web que le permite interactuar con el ejecutable de línea de comandos de IDT y los archivos de configuración relacionados. Puede usar la IU de IDT-FreeRTOS para crear una nueva configuración para ejecutar pruebas de IDT o para modificar una configuración existente. También puede usar la IU para invocar el ejecutable de IDT y ejecutar pruebas.

La IU de IDT-FreeRTOS ofrece las siguientes funciones:

- Simplificar la configuración de los archivos de configuración para las pruebas de IDT-FreeRTOS.
- Simplificar el uso de IDT-FreeRTOS para ejecutar pruebas de calificación.

Para obtener información sobre cómo utilizar la línea de comandos para ejecutar pruebas de calificación, consulte Preparación para probar su placa de microcontrolador por primera vez.

En esta sección se describen los requisitos previos de la IU de IDT-FreeRTOS y cómo empezar a ejecutar las pruebas de calificación en la IU.

Temas

Requisitos previos

En esta sección se describen los requisitos previos para probar los microcontroladores con AWS IoT Device Tester.

Temas

Uso de un navegador web compatible

La IU de IDT-FreeRTOS es compatible con los siguientes navegadores web.

Navegador	Versión
Google Chrome	Tres últimas versiones principales
Mozilla Firefox	Tres últimas versiones principales

Navegador	Versión
Microsoft Edge	Tres últimas versiones principales
Apple Safari para macOS	Tres últimas versiones principales

Le recomendamos que utilice Google Chrome o Mozilla Firefox para disfrutar de una mejor experiencia.

> **Note**
>
> La IU de IDT-FreeRTOS no es compatible con Microsoft Internet Explorer.

Descarga de FreeRTOS

Puede descargar una versión de FreeRTOS desde GitHubcon el siguiente comando:

```
git clone --branch <FREERTOS_RELEASE_VERSION> --recurse-submodules https://github.com/
aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

donde <FREERTOS_RELEASE_VERSION> es una versión de FreeRTOS (por ejemplo, 202007.00) correspondiente a una versión de IDT incluida en Versiones compatibles de AWS IoT Device Tester para FreeRTOS. Esto garantiza que dispone del código fuente completo, incluidos los submódulos, y que utiliza la versión correcta de IDT para su versión de FreeRTOS, y viceversa.

Windows tiene una limitación de longitud de ruta de 260 caracteres. La estructura de la ruta de FreeRTOS tiene muchos niveles de profundidad por lo que, si utiliza Windows, debe mantener las rutas de los archivos dentro de la limitación de 260 caracteres. Por ejemplo, clone FreeRTOS a `C:\FreeRTOS` en lugar de a `C:\Users\username\programs\projects\myproj\FreeRTOS\`.

Consideraciones para la calificación de LTS (calificación para FreeRTOS que utiliza las bibliotecas LTS)

- Para que su microcontrolador se designe como compatible con versiones basadas en soporte a largo plazo (LTS) de FreeRTOS en el Catálogo de dispositivos de socios de AWS, debe

proporcionar un archivo de manifiesto. Para obtener más información, consulte la Lista de verificación de calificación de FreeRTOS en la Guía de calificación de FreeRTOS.

- Para validar que su microcontrolador es compatible con las versiones basadas en LTS de FreeRTOS y poder enviarlo al Catálogo de dispositivos de socios de AWS, debe utilizar AWS IoT Device Tester (IDT) con el conjunto de pruebas FreeRTOS Qualification (FRQ) versión v1.4.x.
- La compatibilidad para las versiones basadas en LTS de FreeRTOS está limitada a la versión 202012.xx de FreeRTOS.

Descarga de IDT para FreeRTOS

Cada versión de FreeRTOS tiene una versión correspondiente de IDT para FreeRTOS para realizar pruebas de calificación. Descargue la versión apropiada de IDT para FreeRTOS en Versiones compatibles de AWS IoT Device Tester para FreeRTOS.

Extraiga IDT para FreeRTOS en una ubicación del sistema de archivos en la que tenga permisos de lectura y escritura. Dado que Microsoft Windows tiene un límite de caracteres para la longitud de la ruta de acceso, extraiga IDT para FreeRTOS en un directorio raíz, como `C:\` o `D:\`.

> **Note**
>
> Se recomienda extraer el paquete IDT a una unidad local. Si se permite que varios usuarios ejecuten IDT desde una ubicación compartida, como un directorio NFS o una carpeta compartida de una red de Windows, el sistema podría no responder o dañar los datos.

Creación y configuración de una cuenta de AWS

Registro para obtener una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Cómo registrarse en 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.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación luego de completar 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.

Crear un usuario administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

Protección de su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Console como propietario de cuenta, elija Usuario raíz e ingrese el email de su Cuenta de AWS. 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 raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario administrativo

1. Activar IAM Identity Center

   Para conocer las instrucciones, consulte Habilitar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center.
2. En IAM Identity Center, otorga acceso administrativo a un usuario administrativo.

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

Cómo iniciar sesión como usuario administrativo

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

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

Política administrada de AWS IoT Device Tester

La política administrada de `AWSIoTDeviceTesterForFreeRTOSFullAccess` contiene los siguientes permisos para permitir que el comprobador de dispositivos ejecute y recopile métricas:

- `iot-device-tester:SupportedVersion`

  Concede permiso para obtener la lista de versiones de FreeRTOS y versiones del conjunto de pruebas compatibles con IDT, de modo que estén disponibles en la AWS CLI.
- `iot-device-tester:LatestIdt`

  Concede permiso para obtener la versión de AWS IoT Device Tester más reciente que se puede descargar.
- `iot-device-tester:CheckVersion`

  Concede permiso para comprobar que la combinación de producto, conjunto de pruebas y versión de AWS IoT Device Tester es compatible.
- `iot-device-tester:DownloadTestSuite`

  Concede permiso a AWS IoT Device Tester para descargar conjuntos de pruebas.
- `iot-device-tester:SendMetrics`

  Concede permiso para publicar datos de métricas de uso de AWS IoT Device Tester.

Introducción a la IU de IDT-FreeRTOS

En esta sección se muestra cómo utilizar la IU de IDT-FreeRTOS para crear o modificar la configuración y, a continuación, se muestra cómo ejecutar las pruebas.

Temas

- Configurar credenciales de AWS

- Apertura de la IU de IDT-FreeRTOS
- Creación de una nueva configuración
- Modificación de una configuración existente
- Ejecución de pruebas de calificación

Configurar credenciales de AWS

Debe configurar las credenciales para el usuario de AWS que ha creado en Creación y configuración de una cuenta de AWS. Puede especificar sus credenciales de una de las dos formas siguientes:

- En un archivo de credenciales
- Como 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>
```

> **Note**
>
> Si no usa el perfil `default` de AWS, debe especificar el nombre del perfil en la IU de IDT para FreeRTOS. Para obtener más información sobre los perfiles, 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. La IU de IDT-FreeRTOS utiliza 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>
```

Apertura de la IU de IDT-FreeRTOS

Para abrir la IU de IDT-FreeRTOS

1. Descargue una versión de IDT-FreeRTOS compatible y extraiga el archivo descargado en una ubicación de su sistema de archivos en la que tenga permisos de lectura y escritura.
2. Ejecute el siguiente comando para navegar al directorio de instalación de IDT-FreeRTOS:

   ```
   cd devicetester-extract-location/bin
   ```

3. Ejecute el siguiente comando para abrir la IU de IDT-FreeRTOS:

   Linux

   ```
   .devicetestergui_linux_x86-64.exe
   ```

   Windows

   ```
   ./devicetestergui_win_x64-64
   ```

   macOS

   ```
   ./devicetestergui_mac_x86-64
   ```

> **Note**
>
> En Mac, para permitir que el sistema ejecute la IU, vaya a Preferencias del sistema -> Seguridad y privacidad. Cuando ejecute las pruebas, es posible que tenga que hacerlo tres veces más.

La IU de IDT-FreeRTOS se abre en el navegador predeterminado. Para obtener información acerca de los navegadores compatibles, consulte Uso de un navegador web compatible.

Creación de una nueva configuración

Si es la primera vez que lo usa, debe crear una nueva configuración para configurar los archivos de configuración JSON que IDT-FreeRTOS necesita para ejecutar las pruebas. A continuación, puede ejecutar pruebas o modificar la configuración creada.

Para ver ejemplos de los archivos `config.json`, `device.json` y `userdata.json`, consulte Preparación para probar su placa de microcontrolador por primera vez. Para ver un ejemplo del archivo `resource.json` que se utiliza únicamente para ejecutar pruebas de Bluetooth de bajo consumo (BLE), consulte Ejecución de pruebas de Bluetooth de bajo consumo.

Para crear una nueva configuración

1. En la IU de IDT-FreeRTOS, abra el menú de navegación y elija Crear nueva configuración.

   > **Important**
   >
   > Debe configurar sus credenciales de AWS antes de abrir la IU. Si no ha configurado sus credenciales, cierre la ventana del navegador de la IU de IDT-FreeRTOS, siga los pasos que se indican en Configurar credenciales de AWS y, a continuación, vuelva a abrir la IU de IDT-FreeRTOS.

2. Siga las instrucciones del asistente de configuración para introducir los ajustes de configuración de IDT que se utilizan para ejecutar las pruebas de calificación. El asistente configura los siguientes ajustes en los archivos de configuración JSON ubicados en el directorio *`devicetester-extract-location`*`/config`.

- Configuración de AWS - La información de Cuenta de AWS que IDT-FreeRTOS utiliza para crear recursos de AWS durante las pruebas. Estos ajustes se configuran en el archivo `config.json`.
- Repositorio de FreeRTOS - La ruta absoluta al repositorio de FreeRTOS y al código portado, y el tipo de calificación que desea realizar. Estos ajustes se configuran en el archivo `userdata.json`.

  Debe realizar la portabilidad de FreeRTOS a su dispositivo antes de poder ejecutar las pruebas de calificación. Para obtener más información, consulte la Guía de portabilidad de FreeRTOS.
- Build y flash - Los comandos build y flash de su hardware que permiten a IDT crear e instalar las pruebas en su placa automáticamente. Estos ajustes se configuran en el archivo `userdata.json`.
- Dispositivos - La configuración del grupo de dispositivos para los dispositivos que se van a probar. Estos ajustes se configuran en los campos `id` y `sku`, y el bloque `devices` para el grupo de dispositivos en el archivo `device.json`.
- Redes - La configuración para probar la compatibilidad de los dispositivos con la comunicación de red. Estos ajustes se configuran en el bloque `features` del archivo `device.json` y en los bloques `clientWifiConfig` y `testWifiConfig` del archivo `userdata.json`.
- Servidor Echo - Los ajustes de configuración del servidor Echo para las pruebas de sockets seguros. Estos ajustes se configuran en el archivo `userdata.json`.

  Para obtener más información sobre la configuración del servidor echo, consulte https://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html.
- CMake - (Opcional) La configuración para ejecutar las pruebas de funcionalidad de creación de CMake. Esta configuración solo es necesaria si utiliza CMake como sistema de creación. Estos ajustes se configuran en el archivo `userdata.json`.
- BLE - La configuración para ejecutar las pruebas de la funcionalidad Bluetooth de bajo consumo. Estos ajustes se configuran en el bloque `features` del archivo `device.json` y en el archivo `resource.json`.
- OTA - La configuración para ejecutar las pruebas de funcionalidad OTA. Estos ajustes se configuran en el bloque `features` del archivo `device.json` y en el archivo `userdata.json`.

3. En la página Revisar, compruebe la información de configuración.

Cuando termine de revisar la configuración, para ejecutar las pruebas de calificación, elija Ejecutar pruebas.

Modificación de una configuración existente

Si ya ha configurado los archivos de configuración para IDT, puede utilizar la IU de IDT-FreeRTOS para modificar la configuración existente. Asegúrese de que los archivos de configuración existentes estén disponibles en el directorio *devicetester-extract-location*/config.

Para modificar una nueva configuración

1. En la IU de IDT-FreeRTOS, abra el menú de navegación y elija Editar configuración existente.

   El panel de configuración muestra información sobre los ajustes de configuración existentes. Si una configuración es incorrecta o no está disponible, el estado de esa configuración es `Error validating configuration`.
2. Complete los siguientes pasos para modificar un ajuste de configuración existente:

   a. Seleccione el nombre de un ajuste de configuración para abrir su página de ajustes.

   b. Modifique los ajustes y, a continuación, seleccione Guardar para volver a generar el archivo de configuración correspondiente.

Cuando termine de modificar la configuración, compruebe que todas las opciones de configuración pasen la validación. Si el estado de cada parámetro de configuración es `Valid`, puede ejecutar las pruebas de calificación con esta configuración.

Ejecución de pruebas de calificación

Después de crear una configuración para la IU de IDT-FreeRTOS, puede ejecutar las pruebas de calificación.

Para ejecutar pruebas de calificación

1. Valide la configuración.
2. En el menú de navegación, elija Ejecutar pruebas.
3. Seleccione Iniciar pruebas para iniciar la ejecución de la prueba.

IDT-FreeRTOS ejecuta las pruebas de calificación y muestra el resumen de la ejecución de la prueba y cualquier error en la consola del Ejecutor de la prueba. Una vez finalizada la ejecución de la prueba, puede ver los resultados y los registros de la prueba desde las siguientes ubicaciones:

- Los resultados de las pruebas se encuentran en el directorio *devicetester-extract-location*/results/*execution-id*.
- Los registros de las pruebas se encuentran en el directorio *devicetester-extract-location*/results/*execution-id*/logs.

Para obtener más información sobre los resultados de las pruebas, consulte Descripción de los resultados y de los registros.

Ejecución de pruebas de Bluetooth de bajo consumo

En esta sección se describe cómo configurar y ejecutar las pruebas de Bluetooth mediante AWS IoT Device Tester para FreeRTOS. Las pruebas de Bluetooth no son obligatorias para la cualificación principal. Si no desea realizar pruebas en su dispositivo con el soporte de Bluetooth para FreeRTOS, puede omitir esta configuración, asegúrese de establecer la característica BLE de device.json en No.

Requisitos previos

- Siga las instrucciones en Preparación para probar su placa de microcontrolador por primera vez.
- Un Raspberry Pi 4B o 3B+. (necesario para ejecutar la aplicación complementaria de Raspberry Pi BLE)
- Una tarjeta microSD y un adaptador de tarjeta SD para el software de Raspberry Pi.

Configuración de Raspberry Pi

Para probar las capacidades de BLE del dispositivo a prueba (DUT), tiene que tener un Raspberry Pi modelo 4B o 3B+.

Para configurar Raspberry Pi y realizar las pruebas de BLE

1. Descargue una de las imágenes de Yocto que contenga el software necesario para realizar las pruebas.

- Imagen para Raspberry Pi 4B
- Imagen para Raspberry Pi 3B+

> **Note**
>
> La imagen de Yocto solo debe usarse para realizar pruebas con AWS IoT Device Tester para FreeRTOS y no para ningún otro propósito.

2. Instale la imagen de Yocto en la tarjeta SD para Raspberry Pi.

   - Mediante una herramienta de escritura en tarjetas SD comoEtcher, instale el archivo *image-name*`.rpi-sd.img` que ha descargado en la tarjeta SD. Dado que la imagen del sistema operativo es grande, este paso puede tardar un tiempo. A continuación, extraiga la tarjeta SD del equipo e inserte la tarjeta microSD en Raspberry Pi.

3. Configure su Raspberry Pi.

   a. Para la primera operación de inicio, le recomendamos que conecte el Raspberry Pi a un monitor, un teclado y un ratón.

   b. Conecte el Raspberry Pi a una fuente de alimentación micro-USB.

   c. Inicie sesión con las credenciales predeterminadas. En el ID de usuario, introduzca **`root`**. En la contraseña, introduzca **`idtafr`**.

   d. Mediante una conexión Ethernet o Wi-Fi, conecte el Raspberry Pi a su red.

      i. Para conectar el Raspberry Pi mediante Wi-Fi, abra `/etc/wpa_supplicant.conf` en el Raspberry Pi y añada sus credenciales de Wi-Fi a la configuración de `Network`.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
        scan_ssid=1
        ssid="your-wifi-ssid"
        psk="your-wifi-password"
        }
```

ii. Ejecute `ifup wlan0` para iniciar la configuración del wifi. La conexión a su red Wi-Fi puede tardar un minuto.

e. Para una conexión Ethernet, ejecute `ifconfig eth0`. Para una conexión wifi, ejecute `ifconfig wlan0`. Anote la dirección IP, que aparece como `inet addr` en la salida del comando. Necesitará la dirección IP más adelante en este procedimiento.

f. (Opcional) Las pruebas ejecutan comandos en el Raspberry Pi a través de SSH mediante las credenciales predeterminadas de la imagen de Yocto. Para aumentar la seguridad, le recomendamos que establezca una autenticación de clave pública en SSH y desactive SSH mediante contraseña.

i. Cree una clave de SSH mediante el comando `ssh-keygen` de OpenSSL. Si ya tiene un par de claves de SSK en el equipo host, le recomendamos crear uno nuevo para permitir que AWS IoT Device Tester para FreeRTOS pueda iniciar sesión en su Raspberry Pi.

> **Note**
>
> Windows no incluye un cliente SSH instalado. Para obtener más información acerca de cómo instalar un cliente SSH en Windows, consulte Descargar el software SSH.

ii. 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`*.

iii. Cuando se le solicite una frase clave, pulse INTRO para continuar.

iv. Para añadir la clave SSH en su Raspberry Pi y que AWS IoT Device Tester para FreeRTOS pueda iniciar sesión en el dispositivo, utilice el comando `ssh-copy-id` desde el equipo host. Este comando añade su clave pública al archivo `~/.ssh/authorized_keys` del Raspberry Pi.

```
ssh-copy-id root@raspberry-pi-ip-address
```

v. Cuando se le solicite una contraseña, introduzca **`idtafr`**. Esta es la contraseña predeterminada para la imagen de Yocto.

> **Note**
>
> El comando `ssh-copy-id` asume que la clave pública se denomina `id_rsa.pub`. En macOS y Linux, la ubicación predeterminada es `~/.ssh/`. En Windows, la ubicación predeterminada es `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.

vi. Para comprobar si la autenticación de clave pública funciona, ejecute `ssh -i` *`/my/path/myKey`* `root@`*`raspberry-pi-device-ip`*.

Si no se le solicita una contraseña, la autenticación de clave pública funciona.

vii. Compruebe que puede iniciar sesión en su Raspberry Pi mediante una clave pública y, a continuación, desactive SSH mediante contraseña.

A. En el Raspberry Pi, edite el archivo `/etc/ssh/sshd_config`.

B. Establezca el atributo `PasswordAuthentication` en `no`.

C. Guarde y cierre el archivo `sshd_config`.

D. Vuelva a cargar el servidor SSH mediante el comando `/etc/init.d/sshd reload`.

g. Cree un archivo `resource.json`.

i. En el directorio en el que ha extraído AWS IoT Device Tester, cree un archivo llamado `resource.json`.

ii. Añada la siguiente información sobre su Raspberry Pi al archivo y *`rasp-pi-ip-address`*sustitúyala por la dirección IP de su Raspberry Pi.

```
[
    {
        "id": "ble-test-raspberry-pi",
        "features": [
            {"name":"ble", "version":"4.2"}
        ],
        "devices": [
```

```
            {
                "id": "ble-test-raspberry-pi-1",
                "connectivity": {
                    "protocol": "ssh",
                    "ip": "rasp-pi-ip-address"
                }
            }
        ]
    }
]
```

iii. Si ha optado por no utilizar una autenticación de clave pública para SSH, añada lo siguiente a la sección `connectivity` del archivo `resource.json`.

```
"connectivity": {
    "protocol": "ssh",
    "ip": "rasp-pi-ip-address",
    "auth": {
        "method": "password",
        "credentials": {
            "user": "root",
            "password": "idtafr"
        }
    }
}
```

iv. (Opcional) si elige utilizar una autenticación de clave pública para SSH, añada lo siguiente a la sección `connectivity` del archivo `resource.json`.

```
"connectivity": {
    "protocol": "ssh",
    "ip": "rasp-pi-ip-address",
    "auth": {
        "method": "pki",
        "credentials": {
            "user": "root",
            "privKeyPath": "location-of-private-key"
        }
    }
}
```

Configuración del dispositivo FreeRTOS

En el archivo `device.json`, establezca la característica `BLE` en `Yes`. Si comienza con un archivo `device.json` desde antes de que las pruebas de Bluetooth estén disponibles, tiene que añadir la característica de BLE a la matriz de `features`:

```
{
    ...
    "features": [
        {
            "name": "BLE",
            "value": "Yes"
        },
    ...
}
```

Ejecución de las pruebas de BLE

Una vez que active la característica BLE en `device.json`, las pruebas de BLE se ejecutan cuando ejecute `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite` sin especificar un group-id.

Si desea ejecutar las pruebas de BLE de forma independiente, puede especificar el ID de grupo de BLE: `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite --userdata` *`path-to-userdata`*`/userdata.json --group-id FullBLE`.

Para que el desempeño sea más fiable, coloque su Raspberry Pi cerca del dispositivo a prueba (DUT).

Resolución de problemas de las pruebas de BLE

Asegúrese de que ha seguido los pasos que se indican en Preparación para probar su placa de microcontrolador por primera vez. Si se produce un error con las pruebas que no pertenecen a la característica de BLE, es probable que el problema no se deba a la configuración de Bluetooth.

Ejecución del conjunto de calificación de FreeRTOS

Utilice el ejecutable de AWS IoT Device Tester para FreeRTOS para interactuar con IDT para FreeRTOS. Los ejemplos de línea de comandos siguientes le muestran como ejecutar las pruebas de cualificación para un grupo de dispositivos (un conjunto de dispositivos idénticos).

IDT v3.0.0 and later

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --group-id group-id  \
    --pool-id your-device-pool \
    --test-id test-id  \
    --upgrade-test-suite y|n  \
    --update-idt y|n  \
    --update-managed-policy y|n  \
    --userdata userdata.json
```

Ejecuta un conjunto de pruebas en un grupo de dispositivos. El archivo `userdata.json` debe estar ubicado en el directorio *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/`.

> **Note**
>
> Si ejecuta IDT para FreeRTOS en Windows, utilice barras diagonales (/) para especificar la ruta al archivo `userdata.json`.

Utilice el siguiente comando para ejecutar un grupo de prueba específico:

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1.0.1  \
    --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

Los parámetros `suite-id` y `pool-id` son opcionales si está ejecutando un único conjunto de pruebas en un único grupo de dispositivos (es decir, tiene un único grupo de dispositivos definido en el archivo `device.json`).

Utilice el siguiente comando para ejecutar un caso de prueba específico:

```
devicetester_[linux | mac | win_x86-64] run-suite  \
    --group-id group-id  \
    --test-id test-id
```

Puede utilizar el comando `list-test-cases` para ver los casos de prueba en un grupo de pruebas.

Opciones de la línea de comandos de IDT para FreeRTOS

group-id

(Opcional) 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.

pool-id

(Opcional) El grupo de dispositivos que se va a probar. Es necesario si define varios grupos de dispositivos en `device.json`. Si solo tiene un grupo de dispositivos, puede omitir esta opción.

suite-id

(Opcional) La versión del conjunto de pruebas que se va a ejecutar. Si no se especifica, IDT utiliza la versión más reciente del directorio de pruebas del sistema.

> **Note**
>
> A partir de IDT v3.0.0, IDT comprueba en línea los conjuntos de pruebas más recientes. Para obtener más información, consulte Versiones del conjunto de pruebas.

test-id

(Opcional) Las pruebas que se van a ejecutar, como una lista separada por comas. Si se especifica, `group-id` debe especificar un solo grupo.

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mqtt --test-id
 mqtt_test
```

update-idt

(Opcional) Si este parámetro no está establecido y hay disponible una versión más reciente de IDT, se le solicitará que actualice IDT. Si este parámetro está establecido en Y, IDT detendrá

la ejecución de la prueba si detecta que hay disponible una versión más reciente. Si este parámetro está establecido en `N`, IDT continuará con la ejecución de la prueba.

update-managed-policy

(Opcional) Si no utiliza este parámetro e IDT detecta que su política gestionada no lo está up-to-date, se le solicitará que la actualice. Si este parámetro está establecido en`Y`, IDT detendrá la ejecución de la prueba si detecta que su política gestionada no lo está. up-to-date Si este parámetro está establecido en `N`, IDT continuará con la ejecución de la prueba.

upgrade-test-suite

(Opcional) Si no se utiliza, y hay disponible una versión más reciente del conjunto de pruebas, se le pedirá que la descargue. Para ocultar el mensaje, especifique `y` para descargar siempre el conjunto de pruebas más reciente o `n` para utilizar el conjunto de pruebas especificado o la versión más reciente en el sistema.

Example

Ejemplo

Para descargar y utilizar siempre el conjunto de pruebas más reciente, utilice el siguiente comando.

```
devicetester_[linux | mac | win_x86-64] run-suite --userdata userdata file --
group-id group ID --upgrade-test-suite y
```

Para utilizar el conjunto de pruebas más reciente en el sistema, utilice el siguiente comando.

```
devicetester_[linux | mac | win_x86-64] run-suite --userdata userdata file --
group-id group ID --upgrade-test-suite n
```

h

Utilice la opción de ayuda para obtener más información sobre las opciones de `run-suite`.

Example

Ejemplo

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v1.7.0 and earlier

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --pool-id your-device-pool  \
    --userdata userdata.json
```

El archivo `userdata.json` debe estar ubicado en el directorio *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/`.

> **Note**
>
> Si ejecuta IDT para FreeRTOS en Windows, utilice barras diagonales (/) para especificar la ruta al archivo `userdata.json`.

Utilice el siguiente comando para ejecutar un grupo de pruebas específico:

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1 --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

`suite-id` y `pool-id` son opcionales si está ejecutando un único conjunto de pruebas en un único grupo de dispositivos (es decir, tiene un único grupo de dispositivos definido en el archivo `device.json`).

Opciones de la línea de comandos de IDT para FreeRTOS

group-id

(Opcional) Especifica el grupo de prueba.

pool-id

Especifica el grupo de dispositivos que probar. Si solo tiene un grupo de dispositivos, puede omitir esta opción.

suite-id

(Opcional) Especifica el conjunto de pruebas que se va a ejecutar.

Comandos de IDT para FreeRTOS

El comando de IDT para FreeRTOS admite las siguientes operaciones:

IDT v3.0.0 and later

`help`

Enumera información acerca del comando especificado.

`list-groups`

Muestra los grupos de un conjunto determinado.

`list-suites`

Muestra los conjuntos disponibles.

`list-supported-products`

Muestra los productos compatibles y las versiones del conjunto de pruebas.

`list-supported-versions`

Muestra las versiones de FreeRTOS y del conjunto de pruebas compatibles con la versión actual de IDT.

`list-test-cases`

Muestra los casos de prueba de un grupo especificado.

`run-suite`

Ejecuta un conjunto de pruebas en un grupo de dispositivos.

Utilice la opción `--suite-id` para especificar una versión del conjunto de pruebas u omítala para utilizar la versión más reciente en el sistema.

Utilice el `--test-id` para ejecutar un caso de prueba individual.

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mqtt --test-id
 mqtt_test
```

Para obtener una lista completa de opciones, consulte Ejecución del conjunto de calificación de FreeRTOS.

> **Note**
>
> A partir de IDT v3.0.0, IDT comprueba en línea los conjuntos de pruebas más recientes. Para obtener más información, consulte Versiones del conjunto de pruebas.

IDT v1.7.0 and earlier

`help`

Enumera información acerca del comando especificado.

`list-groups`

Muestra los grupos de un conjunto determinado.

`list-suites`

Muestra los conjuntos disponibles.

`run-suite`

Ejecuta un conjunto de pruebas en un grupo de dispositivos.

Prueba de recualificación

A medida que se publican nuevas versiones de pruebas de calificación de IDT para FreeRTOS o se actualizan los paquetes o controladores de dispositivos específicos de su placa, puede utilizar IDT para FreeRTOS para probar las placas de los microcontroladores. En calificaciones posteriores, asegúrese de que dispone de las versiones más recientes de FreeRTOS e IDT para FreeRTOS y ejecute de nuevo las pruebas de calificación.

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, escribe un resumen de ejecución de la prueba en la consola y genera dos informes de prueba. Estos informes se pueden encontrar en

devicetester-extract-location/results/*execution-id*/. Ambos informes capturan los resultados de la ejecución del conjunto de pruebas de cualificación.

awsiotdevicetester_report.xml es el informe de prueba de calificación que envía a AWS para mostrar su dispositivo en el Catálogo de dispositivos de socios de AWS. El informe contiene los componentes siguientes:

- La versión IDT para FreeRTOS.
- La versión de FreeRTOS que se ha probado.
- Las características de FreeRTOS que admite el dispositivo en función de las pruebas superadas.
- El SKU y el nombre de dispositivo especificado en el archivo device.json.
- Las características del dispositivo especificado en el archivo device.json.
- El resumen de agregación de los resultados de casos de prueba.
- Un desglose de los resultados de los casos de prueba por bibliotecas que se probaron en función de las funciones del dispositivo (por ejemplo FullWiFi, FullMQTT, etc.).
- Si esta calificación de FreeRTOS es para la versión 202012.00 que usa bibliotecas LTS.

El FRQ_Report.xml es un informe en formato JUnit XML estándar. Puede integrarlo en las plataformas CI y CD, como Jenkins, Bamboo, etc. El informe contiene los componentes siguientes:

- Un resumen de agregación de los resultados de casos de prueba.
- Un desglose de los resultados de los casos de prueba por bibliotecas que se probaron en función de las características de los dispositivos.

Interpretación de los resultados de IDT para FreeRTOS

La sección del informe en awsiotdevicetester_report.xml o FRQ_Report.xml muestra los resultados de las pruebas que se ejecutan.

La primera etiqueta XML <testsuites> contiene el resumen general de la ejecución de las pruebas. Por ejemplo:

```
<testsuites name="FRQ results" time="5633" tests="184" 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 casos de prueba ejecutados.

`failures`

El número de casos de prueba que se ejecutaron, pero que no se superaron.

`errors`

El número de casos de prueba que IDT para FreeRTOS no ha podido ejecutar.

`disabled`

Este atributo no se utiliza y se puede omitir.

Si no hay errores de caso de prueba, el dispositivo cumple los requisitos técnicos para ejecutar FreeRTOS y puede interoperar con servicios de AWS IoT. Si decide mostrar su dispositivo en el Catálogo de dispositivos de socios de AWS, puede utilizar este informe como prueba de calificación.

Si se producen errores en el caso de prueba, puede identificar el caso de prueba fallido revisando las etiquetas XML `<testsuites>`. Las etiquetas XML `<testsuite>` dentro de la etiqueta `<testsuites>` muestran el resumen del resultado de caso de prueba de un grupo de prueba.

```
<testsuite name="FullMQTT" package="" tests="16" failures="0" time="76"
disabled="0" errors="0" skipped="0">
```

El formato es similar a la etiqueta `<testsuites>`, pero con un atributo denominado `skipped` que no se utiliza y que se puede pasar por alto. Dentro de cada etiqueta XML `<testsuite>`, hay etiquetas `<testcase>` para cada uno de los casos de prueba ejecutados para un grupo de prueba. Por ejemplo:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase"
attempts="1"></testcase>
```

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.

`sdk`

Si ejecutó IDT con un SDK, este bloque contiene el nombre y la versión del SDK. Si no ejecutó IDT con un SDK, este bloque contiene:

```
<sdk>
    <name>N/A</vame>
    <version>N/A</version>
</sdk>
```

`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="core-freertos" value="not-supported" type="required"></feature>
```

Las características marcadas como `optional` no son necesarias para la cualificación. Los siguientes fragmentos muestran características opcionales:

```
<feature name="ota-dataplane-mqtt" value="not-supported" type="optional"></feature>
<feature name="ota-dataplane-http" value="not-supported" type="optional"></feature>
```

Si no hay errores de pruebas para las características requeridas, el dispositivo cumple los requisitos técnicos para ejecutar FreeRTOS y puede interoperar con servicios de AWS IoT. Si quiere mostrar su dispositivo en el Catálogo de dispositivos de socios de AWS, puede utilizar este informe como prueba de calificación.

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="FreeRTOSVersion" package="" tests="1" failures="1" time="2"
 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="FreeRTOSVersion" name="FreeRTOSVersion"></testcase>
```

`lts`

Verdadero si reúne los requisitos para una versión de FreeRTOS que usa bibliotecas LTS, falso en caso contrario.

Atributos que se utilizan en la etiqueta **`<testcase>`**

`name`

El nombre del caso de prueba.

`attempts`

Las veces que IDT para FreeRTOS ha ejecutado la 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">
    <failure type="Failure">Reason for the test case failure</failure>
    <error>Reason for the test case execution error</error>
</testcase>
```

Para obtener más información, consulte Solución de problemas.

Visualización de registros de

Encontrará los registros que IDT para FreeRTOS genera a partir de la ejecución de la prueba en *devicetester-extract-location*`/results/`*execution-id*`/logs`. Se generan dos conjuntos de registros:

`test_manager.log`

Contiene los registros generados a partir de IDT para FreeRTOS (por ejemplo, configuración relacionada con los registros y generación de informes).

***`test_group_id`*`__`*`test_case_id`*`.log`** (por ejemplo, **`FullMQTT__Full_MQTT.log`**)

El archivo de registro de un caso de prueba, incluida la salida del dispositivo que se está probando. El nombre que se asigna al archivo de registro depende del grupo de prueba y del caso de prueba ejecutado.

Uso de IDT para desarrollar y ejecutar sus propios conjuntos de pruebas

A partir de la versión v4.0.0 de IDT, IDT para FreeRTOS combina una configuración estandarizada y un formato de resultados con un entorno de conjuntos de pruebas que le permite desarrollar conjuntos de pruebas personalizados para sus dispositivos y el software de los dispositivos. Puede añadir pruebas personalizadas para su propia validación interna o proporcionárselas a sus clientes para que verifiquen 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 una lógica de prueba personalizada para el dispositivo que desea probar.
- Proporcione a IDT sus conjuntos de pruebas personalizados para los ejecutores de las pruebas. Incluya información sobre configuraciones de ajustes específicas de los conjuntos de pruebas.

Para ejecutar conjuntos de pruebas personalizados

- Configure el dispositivo que desea probar.
- Implemente las configuraciones de los ajustes que requieran los conjuntos de pruebas que desee usar.
- 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.

Descarga de la versión más reciente de AWS IoT Device Tester para FreeRTOS

Descargue la versión más reciente 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 un conjunto de pruebas

Los conjuntos de pruebas se componen de tres tipos de archivos:

- Archivos de configuración que proporcionan a IDT información sobre cómo ejecutar el conjunto de pruebas.
- Archivos ejecutables de prueba que IDT utiliza para ejecutar casos de prueba.
- Archivos adicionales necesarios para ejecutar las pruebas.

Complete los siguientes pasos básicos para crear pruebas de IDT personalizadas:

1. Cree archivos de configuración 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 las pruebas ejecuten el conjunto de pruebas.
4. Compruebe que IDT pueda ejecutar su conjunto de pruebas y generar 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: Creación y ejecución del conjunto de pruebas de IDT de muestra.

Para empezar a crear un conjunto de pruebas personalizado en Python, consulte Tutorial: Desarrollo de un conjunto de pruebas de IDT sencillo.

Tutorial: Creación y ejecución del conjunto de pruebas de IDT de muestra

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 muestra para saber cómo puede usar AWS IoT Device Tester para FreeRTOS para ejecutar conjuntos de pruebas personalizados. Aunque en este tutorial se usa SSH, le resultará útil aprender a usar AWS IoT Device Tester con dispositivos FreeRTOS.

En este tutorial, debe completar los siguientes pasos:

1. Creación del conjunto de pruebas de muestra
2. Uso de 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 AWS IoT Device Tester
  - Python 3.7 o posterior

    Para comprobar la versión de Python instalada en el equipo, 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 un terminal de Powershell para establecer `python3` como alias para el comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

Si no se devuelve información sobre la versión o si la versión es inferior a 3.7, siga las instrucciones que se indican en Descarga de Python) para instalar Python 3.7 o superior. 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. Asegúrese de que tiene configurado SSH en el Raspberry Pi para conectarse de forma remota.

Configuración de 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`, indique la siguiente información:

`id`

Un identificador único definido por el usuario para el dispositivo.

`connectivity.ip`

La dirección IP del 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 que se utiliza para iniciar sesión en el dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`devices.connectivity.auth.credentials.password`

La contraseña que se utiliza para iniciar sesión en el 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`

Creación del 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 creación proporcionados. El siguiente árbol de directorios muestra la ubicación de estos archivos de muestra:

```
<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 de configuración de pruebas.

> **Note**
>
> El conjunto de pruebas de muestra incluye el código fuente de Python. No incluya información confidencial en el código del conjunto de pruebas.

Siguiente paso: Uso de IDT para ejecutar el conjunto de pruebas de muestra que creó.

Uso de IDT para ejecutar el conjunto de pruebas de muestra

Para ejecutar el conjunto de pruebas de muestra, 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 AWS 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 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 con la ejecución 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 al dispositivo a través de SSH desde el equipo host.

Tutorial: Desarrollo de un conjunto de pruebas de IDT sencillo

Un conjunto de pruebas combina lo siguiente:

- Ejecutables de prueba que contienen la lógica de la prueba

- Archivos de configuración que describen el conjunto de pruebas

En este tutorial se muestra cómo usar IDT para FreeRTOS para desarrollar un conjunto de pruebas de Python que contenga un único caso de prueba. Aunque en este tutorial se usa SSH, le resultará útil aprender a usar AWS IoT Device Tester con dispositivos FreeRTOS.

En este tutorial, debe completar los siguientes pasos:

1. Creación de un directorio del conjunto de pruebas
2. Creación de archivos de configuración
3. Creación del ejecutable del caso de prueba
4. Ejecución del conjunto de pruebas

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Requisitos del equipo host
  - Última versión de AWS IoT Device Tester
  - Python 3.7 o posterior

    Para comprobar la versión de Python instalada en el equipo, 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 un terminal de Powershell para establecer `python3` como alias para el comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Si no se devuelve información sobre la versión o si la versión es inferior a 3.7, siga las instrucciones que se indican en Descarga de Python) para instalar Python 3.7 o superior. 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. Asegúrese de que tiene configurado SSH en el Raspberry Pi para conectarse de forma remota.

Creación de 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 pruebas. 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
```

Creación de archivos de configuración

El conjunto de pruebas debe contener los siguientes archivos de configuración necesarios:

Archivos necesarios

`suite.json`

Contiene información sobre el conjunto de pruebas. Consulte Configuración de suite.json.

`group.json`

Contiene información sobre un grupo de pruebas. Debe crear un archivo `group.json` para cada grupo de pruebas del conjunto de pruebas. Consulte Configuración de group.json.

`test.json`

Contiene información sobre un caso de prueba. Debe crear un archivo `test.json` para cada caso de prueba de su conjunto de pruebas. Consulte Configuración de test.json.

1. En la carpeta `MyTestSuite_1.0.0/suite`, cree un archivo `suite.json` con la estructura siguiente:

```
{
    "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 estructura siguiente:

```
{
    "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 estructura siguiente:

```
{
    "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 de directorios de la carpeta `MyTestSuite_1.0.0` debe ser similar al siguiente:

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Obtención del SDK de cliente de IDT

Utilice 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 de 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'
```

Creación del 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()
```

Configuración de 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`, indique la siguiente información:

`id`

Un identificador único definido por el usuario para el dispositivo.

`connectivity.ip`

La dirección IP del 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 que se utiliza para iniciar sesión en el dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`devices.connectivity.auth.credentials.password`

La contraseña que se utiliza para iniciar sesión en el 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`

Ejecución del conjunto de pruebas

Después de crear el conjunto de pruebas, querrá asegurarse de que funciona como se espera. Siga los pasos que se indican a continuación para ejecutar el conjunto de pruebas con su grupo de dispositivos existente.

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 el 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 AWS 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 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 con la ejecución de la prueba. Antes de consultar los registros de errores, compruebe lo siguiente:

- El SDK del cliente IDT se encuentra en la carpeta correcta, tal y como se describe en este paso.
- Cumple 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 al dispositivo a través de SSH desde el equipo host.

Creación de archivos de configuración para el conjunto de pruebas de IDT

En esta sección se describen los formatos en los que se crean los archivos de configuración que se incluyen al escribir un conjunto de pruebas personalizado.

Archivos de configuración necesarios

`suite.json`

Contiene información sobre el conjunto de pruebas. Consulte Configuración de suite.json.

`group.json`

Contiene información sobre un grupo de pruebas. Debe crear un archivo `group.json` para cada grupo de pruebas del conjunto de pruebas. Consulte Configuración de group.json.

`test.json`

Contiene información sobre un caso de prueba. Debe crear un archivo `test.json` para cada caso de prueba de su conjunto de pruebas. Consulte Configuración de test.json.

Archivos de configuración opcionales

`test_orchestrator.yaml` o **`state_machine.json`**

Define cómo se ejecutan las pruebas cuando IDT ejecuta el conjunto de pruebas. Consulte Configuración de test_orchestrator.yaml.

> **Note**
>
> A partir de la versión 4.5.2 de IDT, se utiliza el archivo `test_orchestrator.yaml` para definir el flujo de trabajo de las pruebas. En las versiones anteriores de IDT, se utiliza el archivo `state_machine.json`. Para obtener más información sobre la máquina de estados, consulte Configuración de la máquina de estados de IDT.

`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 está presente en el archivo `device.json`. Consulte Configuración de userdata_schema.json.

Los archivos de configuración se colocan en su *<custom-test-suite-folder>*, tal y como se muestra aquí.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configuración de 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 archivo *<custom-test-suite-folder>*`/suite/suite.json`:

```
{
    "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 ID 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 conjunto también deben cumplir los siguientes requisitos:

- *`<suite-name>`* no puede contener guiones bajos.
- *`<suite-version>`* se indica como *`x.x.x`*, donde x es un número.

El ID 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 las pruebas.

`details`

Una descripción corta de la finalidad del conjunto de pruebas.

`userDataRequired`

Define si los ejecutores de las 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 establecer para este conjunto de pruebas.

`environmentVariables.key`

El nombre de la variable de entorno.

`environmentVariables.value`

El valor de la variable de entorno.

Configuración de group.json

El archivo `group.json` define si un grupo de pruebas es obligatorio u opcional. Utilice la siguiente plantilla para configurar el archivo *<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json`:

```
{
    "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 ID 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 ID se utiliza en los informes de prueba generados por IDT.

`title`

Un nombre descriptivo para el grupo de prueba. El nombre se muestra en la CLI de IDT para los ejecutores de las pruebas.

`details`

Una descripción corta de la finalidad del grupo de pruebas.

`optional`

Opcional. Establézcalo en `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`.

Configuración de test.json

El `test.json` archivo determina los ejecutables del caso de prueba y las variables de entorno que utiliza un caso de prueba. Para obtener más información sobre cómo crear ejecutables de casos de prueba, consulte Creación de ejecutables de casos de prueba de IDT.

Utilice la siguiente plantilla para configurar el archivo *<custom-test-suite-folder>*`/suite/`*<test-group>*`/`*<test-case>*`/test.json`:

```
{
    "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 ID único definido por el usuario para el caso de prueba. El valor de `id` debe coincidir con el nombre de la carpeta del caso de prueba en la que se encuentra el archivo `test.json` y no debe contener guiones bajos (_). El ID se utiliza en los informes de prueba generados por IDT.

`title`

Un nombre descriptivo para el caso de prueba. El nombre se muestra en la CLI de IDT para los ejecutores de las pruebas.

`details`

Una breve descripción de la finalidad del caso de prueba.

`requireDUT`

Opcional. Establézcalo en `true` si se requiere un dispositivo para ejecutar esta prueba; de lo contrario, establézcalo en `false`. El valor predeterminado es `true`. Los ejecutores de las pruebas 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 asignará al dispositivo de recursos cuando se ejecute esta prueba.

`requiredResources.features`

Una matriz 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 se coteja con el nombre de la característica que proporciona el ejecutor de las pruebas en el archivo `resource.json`.

`requiredResources.features.version`

Opcional. La versión de la característica. Este valor se coteja con la versión de la característica proporcionada por el ejecutor de las pruebas en el archivo `resource.json`. Si no se proporciona una versión, la característica no se comprueba. Si no se necesita un número de versión 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 establecer este valor, consulte Creación de 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 definidas para este caso de prueba.

`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.

Configuración de test_orchestrator.yaml

Un orquestador de pruebas 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 un orquestador de pruebas definido por el usuario, IDT generará un orquestador de pruebas para usted.

El orquestador de pruebas predeterminado 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 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.

Para obtener más información sobre cómo funciona el orquestador de pruebas, consulte Configuración del orquestador de pruebas de IDT.

Configuración de userdata_schema.json

El archivo `userdata_schema.json` determina el esquema en el que los ejecutores de las pruebas proporcionan los datos de usuario. 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 se incluye en el conjunto de pruebas.

Para indicar que los ejecutores de pruebas deben proporcionar un archivo `userdata.json`:

1. En el archivo `suite.json`, establezca `userDataRequired` en `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 de 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 de IDT como en el contexto del orquestador de pruebas.

Configuración del orquestador de pruebas de IDT

A partir de la versión 4.5.2 de IDT, IDT incluye un nuevo componente de orquestador de pruebas. El orquestador de pruebas es un componente de IDT que controla el flujo de ejecución del conjunto de pruebas y genera el informe de prueba una vez que IDT termina de ejecutar todas las pruebas. El orquestador de pruebas determina la selección de las pruebas y el orden en que se ejecutan en función de las reglas definidas por el usuario.

Si su conjunto de pruebas no incluye un orquestador de pruebas definido por el usuario, IDT generará un orquestador de pruebas para usted.

El orquestador de pruebas predeterminado 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 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.

El orquestador de pruebas reemplaza a la máquina de estados de IDT. Le recomendamos utilizar el orquestador de pruebas para desarrollar sus conjuntos de pruebas en lugar de la máquina de estados de IDT. El orquestador de pruebas ofrece las siguientes funciones mejoradas:

- Utiliza un formato declarativo en comparación con el formato imperativo que utiliza la máquina de estados de IDT. Esto le permite especificar qué pruebas desea ejecutar y cuándo quiere ejecutarlas.
- Administra la gestión de grupos específicos, la generación de informes, la gestión de errores y el seguimiento de los resultados para que no tenga que gestionar estas acciones manualmente.
- Utiliza el formato YAML, que admite comentarios de forma predeterminada.
- Requiere un 80 por ciento menos de espacio en disco que el orquestador de pruebas para definir el mismo flujo de trabajo.
- Añade una validación previa a las pruebas para comprobar que la definición del flujo de trabajo no contiene identificadores de prueba incorrectos o dependencias circulares.

Formato del orquestador de pruebas

Puede utilizar la siguiente plantilla para configurar su propio archivo *custom-test-suite-folder*/suite/test_orchestrator.yaml:

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Aliases`

Opcional. Cadenas definidas por el usuario que se asignan a expresiones de contexto. Los alias le permiten generar nombres descriptivos para identificar las expresiones de contexto en la configuración de su orquestador de pruebas. Esto resulta especialmente útil si está creando expresiones contextuales complejas o expresiones que utiliza en varios lugares.

Puede usar expresiones de contexto para almacenar consultas de contexto que le permitan acceder a los datos de otras configuraciones de IDT. Para obtener más información, consulte Acceso a los datos del contexto.

Example

Ejemplo

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

Opcional. Una lista de condiciones y los casos de prueba correspondientes que se ejecutan cuando se cumple cada condición. Cada condición puede tener varios casos de prueba; sin embargo, puede asignar un caso de prueba determinado a una sola condición.

De forma predeterminada, IDT ejecuta cualquier caso de prueba que no esté asignado a una condición de esta lista. Si no especifica esta sección, IDT ejecuta todos los grupos de pruebas del conjunto de pruebas.

Cada elemento de la lista `ConditionalTests` incluye los siguientes parámetros:

`Condition`

Una expresión de contexto que se evalúa como un valor booleano. Si el valor evaluado es verdadero, IDT ejecuta los casos de prueba que se especifican en el parámetro `Tests`.

`Tests`

La lista de descriptores de prueba.

Cada descriptor de prueba usa el ID del grupo de pruebas y uno o más ID de casos de prueba para identificar las pruebas individuales que se van a ejecutar en un grupo de pruebas específico. El descriptor de la prueba utiliza el siguiente formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example

Ejemplo

En el siguiente ejemplo se utilizan expresiones de contexto genéricas que puede definir como `Aliases`.

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

En función de las condiciones definidas, IDT selecciona los grupos de prueba de la siguiente manera:

- Si `Condition1` es verdadero, IDT ejecuta las pruebas de los grupos de pruebas A, B y C.
- Si `Condition2` es verdadero, IDT ejecuta las pruebas de los grupos de pruebas C y D.

`Order`

Opcional. El orden en que se deben ejecutar las pruebas. El orden de las pruebas se especifica a nivel de grupo de pruebas. Si no especifica esta sección, IDT ejecuta todos los grupos de pruebas aplicables en un orden aleatorio. El valor de `Order` es una lista de listas de descriptores de grupos. Cualquier grupo de pruebas que no incluya en la lista `Order` se puede ejecutar en paralelo con cualquier otro grupo de pruebas de la lista.

Cada lista de descriptores de grupo contiene uno o más descriptores de grupo e identifica el orden en el que se deben ejecutar los grupos que se especifican en cada descriptor. Puede utilizar los siguientes formatos para definir descriptores de grupos individuales:

- *group-id* - El ID de grupo de un grupo de pruebas existente.
- [*group-id*, *group-id*] - Lista de grupos de pruebas que se pueden ejecutar en cualquier orden relativo.
- "*" - Comodín. Esto equivale a la lista de todos los grupos de pruebas que aún no están especificados en la lista de descriptores de grupos actual.

El valor de `Order` también debe cumplir los siguientes requisitos:

- Los ID de los grupos de pruebas que especifique en un descriptor de grupo deben existir en su conjunto de pruebas.
- Cada lista de descriptores de grupos debe incluir al menos un grupo de pruebas.
- Cada lista de descriptores de grupo debe contener identificadores de grupo únicos. No puede repetir el ID de un grupo de pruebas dentro de los descriptores de grupos individuales.
- Una lista de descriptores de grupo puede tener como máximo un descriptor de grupo comodín. El descriptor de grupo comodín debe ser el primer o el último elemento de la lista.

Example

Ejemplo

En el caso de un conjunto de pruebas que contiene los grupos de pruebas A, B, C, D y E, en la siguiente lista de ejemplos se muestran diferentes formas de especificar que IDT debe ejecutar primero el grupo de pruebas A, después el grupo de pruebas B y, por último, ejecutar los grupos de pruebas C, D y E en cualquier orden.

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B
  ```
  ```

```
    - - B
      - C

    - - B
      - D

    - - B
      - E
```

`Features`

Opcional. La lista de características del producto que desea que IDT añada al archivo `awsiotdevicetester_report.xml`. Si no especifica esta sección, IDT no añadirá ninguna característica del producto al informe.

Una característica del producto es información definida por el usuario sobre los criterios específicos que puede cumplir un dispositivo. Por ejemplo, la característica del producto MQTT puede indicar que el dispositivo publica los mensajes MQTT correctamente. En `awsiotdevicetester_report.xml`, 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.

Cada elemento de la lista `Features` incluye los siguientes parámetros:

`Name`

El nombre de la característica.

`Value`

Opcional. El valor personalizado que desea utilizar en el informe en lugar de `supported`. Si no se especifica este valor, IDT establece el valor de la característica en `supported` o `not-supported`, en función de los resultados de las pruebas. Si prueba la misma característica con diferentes condiciones, puede utilizar un valor personalizado para cada instancia de esa característica en la lista `Features` e IDT concatena los valores de la característica para las condiciones admitidas. Para obtener más información, consulte

`Condition`

Una expresión de contexto que se evalúa como un valor booleano. Si el valor evaluado es verdadero, IDT añade la característica al informe de la prueba una vez que termine de

ejecutar el conjunto de pruebas. Si el valor evaluado es falso, la prueba no se incluye en el informe.

`Tests`

Opcional. La lista de descriptores de prueba. Para que la característica sea compatible, se deben superar todas las pruebas especificadas en esta lista.

Cada descriptor de prueba de esta lista usa el ID del grupo de pruebas y uno o más ID de casos de prueba para identificar las pruebas individuales que se van a ejecutar en un grupo de pruebas específico. El descriptor de la prueba utiliza el siguiente formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Debe especificar `Tests` o `OneOfTests` para cada característica de la lista `Features`.

`OneOfTests`

Opcional. La lista de descriptores de prueba. Para que la característica sea compatible, se debe superar al menos una de las pruebas especificadas en esta lista.

Cada descriptor de prueba de esta lista usa el ID del grupo de pruebas y uno o más ID de casos de prueba para identificar las pruebas individuales que se van a ejecutar en un grupo de pruebas específico. El descriptor de la prueba utiliza el siguiente formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Debe especificar `Tests` o `OneOfTests` para cada característica de la lista `Features`.

`IsRequired`

El valor booleano que define si la característica es obligatoria en el informe de prueba. El valor predeterminado es `false`.

Contexto del orquestador de pruebas

El contexto del orquestador de pruebas es un documento JSON de solo lectura que contiene datos que están disponibles para el orquestador de pruebas durante la ejecución. Solo se puede acceder

al contexto del orquestador de pruebas desde el orquestador de pruebas 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 necesario ejecutar una prueba específica.

El contexto del orquestador de pruebas utiliza el siguiente formato:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`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`

Información en el archivo `config.json`.

Puede consultar el contexto mediante la notación JSONPath. La sintaxis de las consultas JSONPath en las definiciones de estado es `{{query}}`. Al acceder a los datos desde el contexto del orquestador de pruebas, asegúrese de que cada valor se evalúe como una cadena, un número o un booleano.

Para obtener más información sobre cómo utilizar la notación JSONPath para acceder a los datos desde el contexto, consulte Uso del contexto de IDT.

Configuración de la máquina de estados de IDT

> **Important**
>
> A partir de la versión 4.5.2 de IDT, esta máquina de estados está obsoleta. Le recomendamos encarecidamente que utilice el nuevo orquestador de pruebas. Para obtener más información, consulte Configuración del orquestador de pruebas de 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 la generará. 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 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 estados 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 a ese estado.
- Cada estado define la regla de transición para el siguiente estado.
- El estado final debe ser `Succeed` o `Fail`.

Formato de las máquinas de estados

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 estados.

`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`

Objeto que asigna los nombres de estado definidos por el usuario a estados de IDT válidos. Cada objeto States.*`state-name`* contiene la definición de un estado válido asignado a *`state-name`*.

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

en el nivel de caso de prueba. Sin embargo, le recomendamos que configure las reglas de transición de estado en el nivel de grupo de pruebas en lugar de en el nivel del caso de prueba, a menos que sea absolutamente necesario.

Definiciones de estado

- RunTask
- Choice
- Parallel
- AddProductFeatures
- 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. Una matriz 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 especifica `TestGroup` y `TestCases`, IDT ejecuta los casos de prueba especificados del grupo de pruebas.
- Cuando se especifica `TestCases`, pero no se especifica `TestGroup`, IDT ejecuta los casos de prueba especificados.
- Cuando se especifica `TestGroup`, pero no se especifica `TestCases`, IDT ejecuta todos los casos de prueba del grupo de pruebas especificado.
- Si no se especifica `TestGroup` ni `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 ejecutores de las pruebas, debe incluir los 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 “Habilitación de comandos de 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 defina en `ResultVar` como `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 ID de grupo de pruebas sin especificar los ID 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 o identificadores de casos de prueba especificados 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`.

Choice

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` como `true`.

`FallthroughOnError`

Opcional. Especifica el comportamiento cuando el estado encuentra un error al evaluar las expresiones. Establézcalo en `true` si desea omitir una expresión si la evaluación genera un error. Si ninguna expresión coincide, la máquina de estados pasa al estado `Default`. Si no se especifica el valor `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 cadena de expresión que se evalúa 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 la máquina 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 establecer `FallthroughOnError` en `false`. Sin embargo, le recomendamos que establezca `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`

Una matriz de definiciones de máquinas de estados para ejecutar. Cada definición de máquina de estados debe contener sus propios estados `StartAt`, `Succeed` y `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 estados de la rama comparte el mismo contexto de máquina de estados, establecer variables en una rama y, a continuación, leer esas variables desde otra rama podría provocar un comportamiento inesperado.

El estado `Parallel` pasa al siguiente estado solo después de ejecutar todas las máquinas de estados 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 estados de la rama como la máquina de estados principal pasen al estado `Fail` para gestionar los errores de ejecución.

Como las máquinas de estados de la rama no transmiten los errores de ejecución a la máquina de estados principal, no puede usar un bloque `Catch` para gestionar los errores de ejecución en las máquinas de estados de la rama. En su lugar, utilice el valor `hasExecutionErrors` en el contexto de la máquina de estados compartida. Para ver un ejemplo de cómo funciona, consulte Ejemplo de máquina de estados: ejecutar dos grupos de prueba en paralelo.

AddProductFeatures

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 del producto `MQTT` 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`

Una matriz 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 pruebas. Para que la característica sea compatible, deben superan la prueba todas las pruebas de cada grupo de pruebas especificado.

`OneOfGroups`

Opcional. Una matriz de los ID de los grupos de pruebas. Para que la función 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:

- Para que la característica sea compatible, deben superan la prueba todos los casos de prueba especificados.
- `Groups` debe contener solo un ID de grupo de pruebas.
- `OneOfGroups` no debe especificarse.

`IsRequired`

Opcional. Establézcalo en `false` para marcar esta característica como una característica opcional en el informe. El valor predeterminado es `true`.

`ExecutionMethods`

Opcional. Una matriz 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` con estado `RunTask` en uno de los siguientes valores:

- Si especificó ID de casos de prueba individuales, establezca `ResultVar` en *`group-id_test-id`*`_passed`.
- Si no especificó ID de casos de prueba individuales, establezca `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 pruebas se determina a partir del valor de la variable *`group-id`*`_passed` en el contexto de la máquina de estados.
- Si especificó ID de 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 de ejecución de `AddProductFeaturesError`. 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 pruebas que se marcarán como seleccionados. Para cada ID de grupo de pruebas de esta matriz, la variable *`group-id`*`_selected` se establece `true` en el contexto. Asegúrese de proporcionar identificadores de grupos de pruebas 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 la máquina 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 necesario ejecutar 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`

Información del 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 prueba específicos para ejecutarlos en lugar de todo el conjunto de pruebas, se crea esta clave y contiene la lista de ID de casos de prueba específicos.

`hasExecutionErrors`

No se cierra cuando se inicia la máquina de estados. Si algún estado detecta errores 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 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` en estado `Choice`.

Cuando accede a los datos del contexto de la máquina de estados, asegúrese de que se cumplan las siguientes condiciones:

- Sus rutas de JSON deben comenzar por `$.`

- Cada valor debe evaluarse como una cadena, un número o un booleano.

Para obtener más información sobre cómo utilizar la notación JSONPath para acceder a los datos desde el contexto, consulte Uso del contexto de IDT.

Errores de ejecución

Los errores de ejecución son errores en la definición de la máquina de estados que esta 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.

Catch

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 genera.

`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 Catch se gestionan secuencialmente hasta que uno coincide. Si los errores no coinciden con los enumerados en los bloques Catch, 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 Error 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 cuándo se produce un error y, a continuación, usar 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 `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 estados de la rama con 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 Catch en su lugar.

Máquinas de estados de ejemplo

En esta sección se proporcionan algunos ejemplos de configuraciones de máquinas de estados.

Ejemplos

- Ejemplo de máquina de estados: ejecutar un solo grupo de pruebas
- Ejemplo de máquina de estados: ejecutar grupos de pruebas seleccionados por el usuario
- Ejemplo de máquina de estados: ejecutar un solo grupo de pruebas con características de productos
- Ejemplo de máquina de estados: ejecutar dos grupos de prueba en paralelo

Ejemplo de máquina de estados: ejecutar un solo grupo de pruebas

Esta máquina de estados:

- Ejecuta el grupo de pruebas con ID `GroupA`, que debe estar presente en el conjunto de un archivo `group.json`.
- Comprueba si hay errores de ejecución y pasa a `Fail` si se encuentra alguno.
- Genera un informe y pasa a `Succeed` si no hay errores y a `Fail` en caso contrario.

```
{
    "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: ejecutar grupos de pruebas seleccionados por el usuario

Esta máquina de estados:

- 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 pruebas:
  - Ejecuta los casos de prueba dentro de los grupos de pruebas 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 sale.
- Si no se seleccionan grupos de pruebas:
  - Ejecuta las pruebas del grupo de pruebas `GroupA`.
  - Genera informes y sale.

```
{
    "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: ejecutar un solo grupo de pruebas con características de productos

Esta máquina de estados:

- Ejecuta el grupo de pruebas `GroupA`.
- Comprueba si hay errores de ejecución y pasa a `Fail` si se encuentra alguno.
- Añade la característica `FeatureThatDependsOnGroupA` al archivo `awsiotdevicetester_report.xml`:
  - Si `GroupA` supera la prueba, la característica se establece en `supported`.
  - La característica no se marca como opcional en el informe.
- Genera un informe y pasa a `Succeed` si no hay errores y a `Fail` en caso contrario.

```
{
    "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: ejecutar dos grupos de prueba en paralelo

Esta máquina de estados:

- Ejecuta los grupos de pruebas `GroupA` y `GroupB` en paralelo. Las variables `ResultVar` almacenadas en el contexto por los estados `RunTask` de las máquinas de estados de la rama están disponibles para el estado `AddProductFeatures`.

- Comprueba si hay errores de ejecución y pasa 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 la rama.
- Agrega características al archivo `awsiotdevicetester_report.xml` en función de los grupos que aprueban
  - Si `GroupA` supera la prueba, la característica se establece en `supported`.
  - La característica no se marca como opcional en el informe.
- Genera un informe y pasa a `Succeed` si no hay errores y a `Fail` en caso contrario.

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` incluyen varias pruebas, es posible que ambos dispositivos se asignen a esas pruebas. Si solo se configura un dispositivo, los grupos de pruebas 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"
        }
    }
}
```

Creación de 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 ejecutable de caso de prueba 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 del 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.

Uso del 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. Actualmente, IDT ofrece los siguientes SDK:

- SDK de cliente de IDT para Python
- SDK de cliente de IDT para Go
- SDK de cliente de IDT para Java

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 con el 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 Uso del contexto de IDT.

Interacción con 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 Uso del contexto de IDT.

`SendResult`

Permite que los conjuntos de pruebas notifiquen los resultados de los casos de prueba a IDT. Debe llamarse a este comando 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 la máquina host.

`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 Uso del 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 de los casos de prueba ejecutables.

Habilitación de comandos de CLI de IDT

El comando `run-suite` de la CLI de IDT 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 que se admita 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 Especificación del 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 anticipadamente en caso de error, consulte Especificación del 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 `run-suite` comando para especificar el siguiente comportamiento de ejecución de la prueba:

- Ejecutar todas las pruebas dentro de los grupos de pruebas especificados.
- Ejecutar una selección de pruebas desde un grupo de pruebas especificado.

Para admitir estos argumentos, la máquina de estados de su conjunto de pruebas debe incluir un conjunto específico de estados `RunTask` y `Choice` en su máquina de estados. Si no utiliza una máquina de estados personalizada, la máquina de estados de IDT predeterminada incluye los estados necesarios y no es necesario que realice ninguna acción adicional. Sin embargo, si utiliza una máquina de estados personalizada, use Ejemplo de máquina de estados: ejecutar grupos de pruebas seleccionados por el usuario como ejemplo para añadir los estados necesarios a su máquina de estados.

Para obtener más información sobre los comandos de la CLI de IDT, consulte Depuración y ejecución de conjuntos de pruebas personalizados.

Escritura de registros de eventos

Mientras se ejecuta la prueba, se envían datos a `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 de 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.

Notificación de 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 Revisión de 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 notificar 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 notifica los resultados a través de la API, IDT busca los resultados de las pruebas en la carpeta de artefactos de la prueba. La ruta a esta carpeta se almacena en la testData.testArtifactsPath indicada 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 la prueba genera 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 la prueba resuma con precisión los resultados de la prueba 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.

Especificación del 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, lo marca indicando 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 error.

Para detectar este evento, puede sondear periódicamente la API para comprobar el valor del booleano `CancellationRequested` en la respuesta de la API `PollForNotifications`. Cuando IDT detecta un fallo y está configurado 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 están ejecutando 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 tiempo de espera agotado. Como práctica recomendada, debe asegurarse de que los casos de prueba realicen las siguientes acciones cuando detecten alguno de estos eventos:

1. Dejar de ejecutar la lógica de prueba normal.
2. Limpiar todos los recursos temporales, como los artefactos de prueba del dispositivo que se está probando.
3. Notificar el resultado de una prueba a IDT, como un fallo o un error en la prueba.
4. Salir.

Uso del 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 de 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 de 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,
        "idtRootPath": <path/to/IDT/root>
    },
    "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 los siguientes campos adicionales:

`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.

`config.idRootPath`

Este valor es un marcador de posición para el valor de ruta absoluto de IDT al configurar el archivo `userdata.json`. Se utiliza en los comandos build y flash.

`device`

Información sobre el dispositivo seleccionado para la ejecución de 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 grupo de dispositivos en el nivel superior definido en el archivo `device.json` para el grupo de dispositivos seleccionado.

`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` en el archivo `test.json`.

`testData.awsCredentials`

Las credenciales de AWS que se utilizan en la prueba para conectarse a la nube de 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`.

Acceso a los datos del contexto

Puede consultar el contexto mediante la notación JSONPath de sus archivos de configuración y de su ejecutable de texto con las API `GetContextValue` y `GetContextString`. La sintaxis de las cadenas JSONPath para acceder al contexto de IDT varía de la siguiente manera:

- En `suite.json` y `test.json`, se usa `{{`*`query`*`}}` Es decir, no utilice el elemento raíz `$.` para iniciar la expresión.
- En `statemachine.json`, se usa `{{$.`*`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 foobar típica:

Operador	Descripción
`$`	El elemento raíz. Como el valor de contexto de nivel superior de IDT es un objeto, se suele utilizar `$.` para iniciar las consultas.
`.childName`	Accede al elemento secundario con el nombre `childName` desde un objeto. Si se aplica a una matriz, genera una nueva matriz con este operador aplicado a cada elemento. El nombre del elemento distingue entre mayúsculas y minúsculas. Por ejemplo, la consulta para acceder al valor `awsRegion` del objeto `config` es `$.config.awsRegion` .

Operador	Descripción
`[start:end]`	Filtra los elementos de una matriz y los recupera desde el índice `start` hasta el índice `end`, ambos incluidos.
`[index1, index2, ... , indexN]`	Filtra los elementos de una matriz y los recupera únicamente de los índices especific ados.
`[?(expr)]`	Filtra los elementos de una matriz con la expresión `expr`. Esta expresión debe evaluarse en un valor booleano.

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 de matriz, booleano o de objeto, este es el único operador binario compatible que puede utilizar.
  - >= (Mayor o igual que)
  - > (Mayor que)
  - =~ (Coincidencia de expresión regular). Para usar este operador en una expresión de filtro, el JSONPath o el valor del lado izquierdo de la expresión debe evaluarse como una cadena y el lado derecho debe ser un valor de patrón que siga la sintaxis RE2.

Puede utilizar consultas JSONPath con el formato {{*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 los archivos `test.json` y `suite.json`, las consultas tienen en cuenta las siguientes consideraciones:

- Debe escribir en minúsculas cada vez que aparezca la clave `devicePool` en la consulta. Es decir, utilizar `devicepool` en su lugar.
- 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 que no se pueda distinguir 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 los archivos `test.json` y `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.

Configuración de 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 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 de 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 inicio de sesión, 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>",
                "pairedResource": "<device-id>", //used for no-op protocol
                "connectivity": {
                    "protocol": "ssh | uart | docker | no-op",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "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 calificados.

> **Note**
>
> Si desea enumerar la placa en el Catálogo de dispositivos de socios de AWS, el SKU que especifique aquí debe coincidir con el SKU que utilice en el proceso de enumeración.

`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 ejecutores de 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 su valor en los niveles de QOS compatibles con su dispositivo. Puede recuperar la información proporcionada desde el contexto de IDT con la consulta `devicePool.features` o desde el contexto de la máquina de estados con la consulta `pool.features`.

`features.name`

El nombre de la característica.

`features.value`

Los valores de la característica admitidos.

`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`

Una matriz de dispositivos en el grupo que se va 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.

`devices.pairedResource`

Un identificador único definido por el usuario para un dispositivo de recursos. Este valor es obligatorio cuando se prueban dispositivos mediante el protocolo de conectividad `no-op`.

`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 admitidos son `ssh` y `uart` para los dispositivos físicos, `docker` para los contenedores de Docker y `no-op` para los dispositivos que no tienen una conexión directa con la máquina host de IDT, pero que requieren un dispositivo de recursos como middleware físico para comunicarse con la máquina host.

En el caso de los dispositivos no operativos, el ID del dispositivo de recursos se configura en `devices.pairedResource`. También debe especificar este ID en el archivo `resource.json`. El dispositivo emparejado debe ser un dispositivo que esté físicamente emparejado con el dispositivo que se está probando. Una vez que IDT identifique y se conecte al dispositivo de recursos emparejado, IDT no se conectará a otros dispositivos de recursos según las funciones descritas en el archivo `test.json`.

`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 que se va a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.publicKeyPath`

Opcional. La ruta completa a la clave pública utilizada para autenticar las conexiones al dispositivo que se está probando. Al especificar la `publicKeyPath`, IDT valida la clave pública del dispositivo cuando establece una conexión SSH con el dispositivo que se está probando. Si no se especifica este valor, IDT crea una conexión SSH, pero no valida la clave pública del dispositivo.

Le recomendamos encarecidamente que especifique la ruta a la clave pública y que utilice un método seguro para obtenerla. En el caso de los clientes SSH estándar basados en la línea de comandos, la clave pública se proporciona en el archivo `known_hosts`. Si especifica un archivo de clave pública independiente, este archivo debe usar el mismo formato que el archivo `known_hosts`, es decir, `ip-address key-type public-key`.

`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 que 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 que se va 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 recuperar `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 para los tipos de dispositivos incorrectos.

(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 archivo `userdata_scheme.json` definido en el conjunto de pruebas. Si es un redactor de pruebas, asegúrese de proporcionar esta información a los usuarios que van a ejecutar los conjuntos de pruebas que escriba.

(Opcional) Configuración de resource.json

El archivo `resource.json` contiene información sobre los dispositivos que se van a utilizar 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, puede usar un dispositivo de recursos para comprobar si el dispositivo se puede conectar correctamente a él. Los dispositivos de recursos son opcionales y puede requerir tantos dispositivos de recursos como necesite. Como redactor de pruebas, utilice el archivo test.json para definir las características del dispositivo de recursos que se requieren para una prueba. A continuación, los ejecutores de pruebas utilizan el archivo `resource.json` para proporcionar un

grupo 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>,
                    "publicKeyPath": "<public-key-path>",
                    "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`

Una matriz de dispositivos en el grupo que se va 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 que se va a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.publicKeyPath`

Opcional. La ruta completa a la clave pública utilizada para autenticar las conexiones al dispositivo que se está probando. Al especificar la `publicKeyPath`, IDT valida la clave pública del dispositivo cuando establece una conexión SSH con el dispositivo que se está probando. Si no se especifica este valor, IDT crea una conexión SSH, pero no valida la clave pública del dispositivo.

Le recomendamos encarecidamente que especifique la ruta a la clave pública y que utilice un método seguro para obtenerla. En el caso de los clientes SSH estándar basados en la línea de comandos, la clave pública se proporciona en el archivo `known_hosts`. Si especifica un archivo de clave pública independiente, este archivo debe usar el mismo formato que el archivo `known_hosts`, es decir, `ip-address key-type public-key`.

`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 que 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 que se va 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 la 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 de 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 que 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 puede 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 de la carpeta *< device-tester-extract-location >*.

`configFiles.root`

La ruta a la carpeta que contiene los archivos de configuración.

`configFiles.device`

La ruta al 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 de 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 IDT utiliza 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`.

Depuración y ejecución de conjuntos de pruebas personalizados

Una vez establecida la configuración requerida, IDT puede ejecutar su conjunto de pruebas. El tiempo de ejecución del conjunto de pruebas completo depende del hardware y de la composición del conjunto de pruebas. Como referencia, se tarda aproximadamente 30 minutos en completar el conjunto de pruebas completo de calificación FreeRTOS en un Raspberry Pi 3B.

Mientras escribe su conjunto de pruebas, puede usar IDT para ejecutarlo en modo de depuración, comprobar el código antes de ejecutarlo o proporcionárselo a los ejecutores de pruebas.

Ejecución de 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 conjuntos de pruebas 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 orquestador de pruebas, 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 el orquestador de pruebas 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 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 de ajustes 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 establecer 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 del conjunto de pruebas y, a continuación, responde a ellas. IDT también genera las variables de entorno que se requieran para el proceso de casos para el SDK de cliente de IDT.
3. En su IDE, utilice la configuración `run` o `debug` para hacer lo siguiente:

   a. Establecer los valores de las variables de entorno generadas por IDT.

   b. Establecer el valor de cualquier variable de entorno o argumento que haya especificado en el archivo `test.json` y `suite.json`.

   c. Establecer 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. En el modo de depuración, IDT no agota el tiempo de espera.

5. Una vez completada la depuración, interrumpa IDT para salir del modo de depuración.

Comandos de la CLI de IDT para ejecutar pruebas

En la sección siguiente se describen los comandos de la CLI de IDT.

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 versiones de FreeRTOS y del conjunto de pruebas de calificación de FreeRTOS disponibles 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 pruebas 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. Estas son 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 pruebas 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 que se van a ejecutar, como una 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. Los ejecutores de las pruebas deben especificar un grupo si tienen 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 ejecutar el conjunto de pruebas. Esto solo es necesario si `userdataRequired` está establecido 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 pruebas en modo de depuración. Para obtener más información, consulte Ejecución de IDT en modo de depuración.

Revisión de los resultados y registros de las pruebas de IDT

En esta sección se describe el formato en 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 mensaje de consola generado por IDT.

```
[INFO] [2000-01-02 03:04:05]: 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 ID único para el proceso de IDT actual. Este ID 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 un mensaje de consola generado por un conjunto de pruebas.

```
[INFO] [2000-01-02 03:04:05]: Hello world! suiteId=MyTestSuitegroupId=myTestGroup
 testCaseId=myTestCase deviceId=my-
deviceexecutionId=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.

`groupId`

El ID del grupo de pruebas que se está ejecutando.

`testCaseId`

El ID del caso de prueba que se está ejecutando.

`deviceId`

Un ID del dispositivo que se está probando y que el caso de prueba está utilizando.

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. En el siguiente ejemplo se muestra un mensaje de resumen de la 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 AWS 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 informe 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 del informe y la clave utilizada para firmarlo.
- El SKU del dispositivo y el grupo de dispositivos especificado en el archivo `device.json`.
- La versión del producto y las características del dispositivo que se han probado.

- El resumen de agregación de los resultados de las pruebas. Esta información es la misma que la que se incluye 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 necesarias 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 han ejecutado 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 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 que se utiliza para mejorar la funcionalidad de IDT. IDT recopila información como la siguiente:

- El ID de la cuenta de AWS utilizada 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 credenciales de AWS y, si tiene almacenadas credenciales de AWS, no configure el archivo `config.json` para acceder a ellas.

Registro para obtener una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Cómo registrarse en 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.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación luego de completar 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.

Crear un usuario administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

Protección de su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Console como propietario de cuenta, elija Usuario raíz e ingrese el email de su Cuenta de AWS. 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 raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario administrativo

1. Activar IAM Identity Center

   Para conocer las instrucciones, consulte Habilitar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center.
2. En IAM Identity Center, otorga acceso administrativo a un usuario administrativo.

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

Cómo iniciar sesión como usuario administrativo

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

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

Para dar acceso, añada permisos a los usuarios, grupos o roles:

- Usuarios y grupos en AWS IAM Identity Center:

  Cree un conjunto de permisos. Siga las instrucciones de Creación de un conjunto de permisos en la Guía del usuario de AWS IAM Identity Center.

- Usuarios administrados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones de Creación de un rol para un proveedor de identidades de terceros (federación) en la Guía del usuario de IAM.

- Usuarios de IAM:
  - Cree un rol que el usuario pueda aceptar. Siga las instrucciones descritas en Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.
  - (No recomendado) Asocie una política directamente a un usuario o añada un usuario a un grupo de usuarios. Siga las instrucciones descritas en Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Proporcionar las credenciales de AWS a IDT

Para permitir que IDT acceda a sus credenciales de AWS y envíe las métricas a AWS, haga lo siguiente:

1. Guarde las credenciales de AWS 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 al archivo `.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.

Versiones del conjunto de pruebas de AWS IoT Device Tester para FreeRTOS

IDT para FreeRTOS organiza los recursos de 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 FreeRTOS.
- Un grupo de pruebas es el conjunto de pruebas individuales relacionadas con una característica particular, como la mensajería BLE y MQTT.

A partir de IDT v3.0.0, los conjuntos de pruebas se versionan utilizando un formato `major.minor.patch` que comienza a partir de 1.0.0. Al descargar IDT, el paquete incluye la versión más reciente del conjunto de pruebas.

Cuando inicia IDT en la interfaz de línea de comandos, IDT comprueba si hay disponible una versión más reciente del conjunto de pruebas. Si es así, le pedirá que se actualice a la nueva versión. Puede optar por actualizar o continuar con sus pruebas actuales.

> **Note**
>
> IDT admite las tres versiones más recientes del conjunto de pruebas para la cualificación. Para obtener más información, consulte Política de compatibilidad para AWS IoT Device Tester para FreeRTOS.

Puede descargar conjuntos de pruebas mediante el comando `upgrade-test-suite`. O bien, puede usar el parámetro opcional `-upgrade-test-suite` *`flag`* cuando inicie IDT, donde la *`marca`* puede ser “y”, para descargar siempre la versión más reciente, o “n” para usar la versión existente.

También puede ejecutar el comando `list-supported-versions` para enumerar las versiones de FreeRTOS y del conjunto de pruebas compatibles con la versión actual de IDT.

Las nuevas pruebas podrían introducir nuevas opciones de configuración de IDT. Si los ajustes son opcionales, IDT se lo notifica y continúa ejecutando las pruebas. Si los ajustes son obligatorios, IDT se lo notifica y deja de ejecutarse. Después de configurar los ajustes, puede continuar ejecutando las pruebas.

Solución de problemas

Cada ejecución del conjunto de pruebas tiene un ID de ejecución único que se utiliza para crear una carpeta llamada `results/`*`execution-id`* en el directorio `results`. Los registros de grupos de pruebas individuales se encuentran en el directorio `results/`*`execution-id`*`/logs`. Utilice la salida de la consola de IDT para FreeRTOS para encontrar el ID de ejecución, el del caso de prueba y el del grupo de pruebas que tiene errores y, a continuación, abra el archivo de registro de ese caso de prueba llamado `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log`. La información que incluye este archivo es la siguiente:

- Salida completa de los comandos Build y Flash.
- Salida de la ejecución de la prueba.
- Más información sobre la salida de la consola de IDT para FreeRTOS.

Le recomendamos que utilice el siguiente flujo de trabajo para solucionar problemas:

1. Si aparece un error que indica que el *`usuario/rol`* no está autorizado para acceder a este recurso, asegúrese de configurar los permisos como se especifica en Creación y configuración de una cuenta de AWS.
2. Lea la salida de la consola para obtener información, como el UUID de ejecución y las tareas que se están ejecutando actualmente.
3. Examine el archivo `FRQ_Report.xml` para ver la lista de errores de cada prueba. Este directorio contiene los registros de ejecución de cada grupo de pruebas.
4. Consulte los archivos de registro de `/results/`*`execution-id`*`/logs`.
5. Investigue alguna de las siguientes áreas de problemas:

    - Configuración del dispositivo, como archivos de configuración JSON en la carpeta `/configs/`.

- Interfaz del dispositivo. Compruebe los registros para determinar qué interfaz produce el error.
- Herramientas de dispositivos. Asegúrese de que las cadenas de herramientas de compilación y actualización del dispositivo estén instaladas y configuradas correctamente.
- Para FRQ 1.x.x, asegúrese de que haya disponible una versión limpia y clonada del código fuente de FreeRTOS. Las versiones de FreeRTOS se etiquetan según la versión de FreeRTOS. Para clonar una versión específica del código, utilice los comandos siguientes:

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

Solución de problemas de configuración del dispositivo

Al utilizar IDT para FreeRTOS, debe tener implementados los archivos de configuración correctos antes de ejecutar el binario. Si obtiene errores de análisis y de configuración, el primer paso debe ser localizar y utilizar una plantilla de configuración adecuada para su entorno. Estas plantillas se encuentran en el directorio *IDT_ROOT*`/configs`.

Si continúa teniendo problemas, consulte el siguiente proceso de depuración.

¿Dónde buscar?

Para empezar, lea la salida de la consola para obtener información, como el UUID de ejecución, que se denomina `execution-id` en esta documentación.

A continuación, examine el archivo `FRQ_Report.xml` del directorio `/results/`*`execution-id`*. Este archivo contiene todos los casos de prueba ejecutados y los fragmentos de error de cada error. Para obtener todos los registros de ejecución, busque el archivo `/results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` de cada caso de prueba.

Códigos de error de IDT

En la siguiente tabla se explican los códigos de error generados por IDT para FreeRTOS:

Código de error	Nombre del código de error	Causa posible	Solución de problemas
201	InvalidInputError	Los campos de `device.json`, `config.json` o `userdata.json` faltan o tienen un formato incorrecto.	Asegúrese de que no falten los campos obligatorios y de que tengan el formato obligatorio en los archivos de la lista. Para obtener más información, consulte Preparación para probar su placa de microcontrolador por primera vez.
202	ValidationError	Los campos de `device.json`, `config.json` o `userdata.json` contienen valores no válidos.	Compruebe el mensaje de error en la parte derecha del código de error del informe: • Región de AWS no válida - Especifique una región de AWS válida en su archivo `config.json`. Para obtener más información sobre las regiones de AWS, consulte Regiones y puntos de conexión. • Credenciales de AWS no válidas - Configure credenciales de AWS válidas

Código de error	Nombre del código de error	Causa posible	Solución de problemas
			en la máquina de pruebas (usando variables de entorno o el archivo de credencia les). Compruebe que el campo de autenticación se haya configura do correctam ente. Para obtener más información, consulte Creación y configuración de una cuenta de AWS.

Código de error	Nombre del código de error	Causa posible	Solución de problemas
203	CopySourceCodeError	No se puede copiar el código fuente de FreeRTOS en el directorio especificado.	Compruebe los siguientes elementos: • Compruebe que se haya especificado una `sourcePath` válida en el archivo `userdata.json` . • Elimine la carpeta `build` del directorio del código fuente de FreeRTOS, si existe. Para obtener más información, consulte Configurar ajustes de Build, Flash y Test. • Windows tiene un límite de caracteres para los nombres de las rutas de los archivos. Si introduce un nombre de ruta de archivo largo, se producirá un error.

Código de error	Nombre del código de error	Causa posible	Solución de problemas
204	BuildSourceError	No se puede compilar el código fuente de FreeRTOS.	Compruebe los siguientes elementos: • Compruebe que la información de `buildTool` del archivo `userdata.json` sea correcta. • Si utiliza `cmake` como herramienta de compilación, asegúrese de que se haya especificado `{{enableTests}}` en el comando `buildTool` . Para obtener más información, consulte Configurar ajustes de Build, Flash y Test. • Si ha extraído IDT para FreeRTOS a una ruta de archivo de su sistema que contiene espacios, por ejemplo, `C:\Users\My Name\Desktop\`, es posible que necesite comillas

Código de error	Nombre del código de error	Causa posible	Solución de problemas
			adicionales dentro de los comandos de compilación para asegurarse de que las rutas se analizan correctamente. Es posible que necesite lo mismo para los comandos flash.
205	FlashOrRunTestError	IDT para FreeRTOS no puede instalar ni ejecutar FreeRTOS en su DUT.	Compruebe que la información de `flashTool` del archivo `userdata.json` sea correcta. Para obtener más información, consulte Configurar ajustes de Build, Flash y Test.
206	StartEchoServerError	IDT para FreeRTOS no puede iniciar el servidor Echo para las pruebas de Wi-Fi o sockets seguros.	Compruebe que los puertos que están configurados en `echoServerConfiguration` del archivo `userdata.json` no estén en uso o bloqueados por la configuración de la red o del firewall.

Depuración de errores de análisis del archivo de configuración

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 para FreeRTOS realiza la validación de 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 sigue teniendo problemas para localizar el error, puede realizar manualmente una validación en su IDE, un editor de texto como Atom, Sublime o a través de una herramienta en línea como JSONLint.

Depuración de errores de análisis de los resultados de las pruebas

Al ejecutar un grupo de pruebas de FreeRTOS-Libraries-Integration-Tests, como FullTransportInterfaceTLS, FullPKCS11_Core, FullPKCS11_Onboard_ECC, FullPKCS11_Onboard_RSA, FullPKCS11_PreProvisioned_ECC, FullPKCS11_PreProvisioned_RSA o OTACore, IDT para FreeRTOS analiza el resultados de la prueba del dispositivo de prueba con la conexión en serie. A veces, las salidas en serie adicionales del dispositivo pueden interferir con el análisis de los resultados de las pruebas.

En el caso mencionado anteriormente, se emiten extraños motivos de fallo en un caso de prueba, como cadenas que se originan en salidas de dispositivos no relacionados. El archivo de registro de casos de prueba de IDT para FreeRTOS (que incluye todas las salidas en serie que IDT para FreeRTOS ha recibido durante la prueba) puede mostrar lo siguiente:

```
<unrelated device output>
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities)<unrelated device output>
<unrelated device output>
 PASS
```

En el ejemplo anterior, la salida del dispositivo no relacionado impide que IDT para FreeRTOS detecte el resultado de la prueba, que es APROBADO.

Compruebe lo siguiente para garantizar que las pruebas sean óptimas.

- Asegúrese de que las macros de registro utilizadas en el dispositivo sean seguras para subprocesos. Consulte Implementación de las macros de registro de la biblioteca para obtener más información.

- Asegúrese de que haya un mínimo de salidas en la conexión en serie durante las pruebas. Las salidas de otros dispositivos pueden ser un problema incluso si las macros de registro son seguras para subprocesos, ya que los resultados de las pruebas se generarán en llamadas independientes durante las pruebas.

Lo ideal es que un registro de casos de prueba de IDT para FreeRTOS muestre un resultado de prueba ininterrumpido como el siguiente:

```
---------STARTING TESTS---------
TEST(Full_OTA_PAL, otaPal_CloseFile_ValidSignature) PASS
TEST(Full_OTA_PAL, otaPal_CloseFile_InvalidSignatureBlockWritten) PASS
-----------------------
2 Tests 0 Failures 0 Ignored
```

Depuración de errores de comprobación de integridad

Si utiliza la versión FRQ 1.x.x de FreeRTOS, se aplican las siguientes comprobaciones de integridad.

Al ejecutar el grupo de pruebas FreeRTOSIntegrity, si detecta errores, asegúrese primero de no haber modificado ninguno de los archivos del directorio *`freertos`*. Si no lo ha hecho y sigue teniendo problemas, asegúrese de que está utilizando la rama correcta. Si ha ejecutado el comando `list-supported-products` de IDT, podrá encontrar qué rama etiquetada del repositorio *`freertos`* debería estar usando.

Si ha clonado la rama etiquetada correcta del repositorio `freertos` y sigue teniendo problemas, asegúrese de haber ejecutado también el comando `submodule update`. El flujo de trabajo de clonación del repositorio `freertos` es el siguiente.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout —init —recursive
```

La lista de archivos que busca el comprobador de integridad se encuentra en el archivo `checksums.json` de su directorio *`freertos`*. Para calificar un puerto de FreeRTOS sin modificar los archivos ni la estructura de carpetas, asegúrese de que no se haya modificado ninguno de los archivos que figuran en las secciones ‘`exhaustive`‘ y ‘`minimal`‘ del archivo `checksums.json`.

Para ejecutarlo con un SDK configurado, compruebe que no se haya modificado ninguno de los archivos de la sección ‘`minimal`‘.

Si ha ejecutado IDT con un SDK y ha modificado algunos archivos del directorio *`freertos`*, asegúrese de configurar correctamente el SDK en el archivo `userdata`. De lo contrario, el comprobador de integridad verificará todos los archivos del directorio *`freertos`*.

Depuración de errores en grupos de pruebas FullWiFi

Si utiliza FRQ 1.x.x, detecta errores en el grupo de pruebas FullWiFi y la prueba “`AFQP_WiFiConnectMultipleAP`” ha fallado, podría deberse a que ambos puntos de acceso no están en la misma subred que el equipo host que ejecuta IDT. Asegúrese de que ambos puntos de acceso estén en la misma subred que el equipo host que ejecuta IDT.

Depuración de error de “ausencia de parámetro obligatorio”

Dado que se están añadiendo nuevas características a IDT para FreeRTOS, 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, en el archivo *`test_group_id`*`__`*`test_case_id`*`.log` del directorio `results/`*`execution-id`*`/logs` se muestran explícitamente todos los parámetros que faltan. IDT para FreeRTOS también valida los esquemas del archivo de configuración JSON para asegurarse de que se ha utilizado la versión más reciente compatible.

Depuración del error de “imposibilidad de iniciar una prueba”

Es posible que vea errores que apuntan a fallos durante el inicio de las pruebas. Dado que hay varias causas posibles, asegúrese de comprobar que las siguientes áreas son correctas:

- Asegúrese de que el nombre del grupo que ha incluido en el comando de ejecución realmente existe. A esto se hace referencia directamente desde el archivo `device.json`.
- Asegúrese de que el dispositivo o dispositivos del grupo tienen parámetros de configuración correctos.

Depuración de un error que indica que “no se encuentran los resultados del inicio de la prueba”

Es posible que vea errores cuando IDT intente analizar los resultados generados por el dispositivo que se está probando. Dado que hay varias causas posibles, asegúrese de comprobar que las siguientes áreas son correctas:

- Asegúrese de que el dispositivo que se está probando tenga una conexión estable con su máquina host. Puede consultar el archivo de registro para ver si hay una prueba que muestre estos errores para ver qué recibe IDT.
- Si utiliza FRQ 1.x.x y el dispositivo que se está probando está conectado a través de una red lenta u otra interfaz, o no ve el indicador “---------STARTING TESTS---------” en el registro de un grupo de pruebas de FreeRTOS junto con las salidas de otros grupos de pruebas de FreeRTOS, puede intentar aumentar el valor de `testStartDelayms` en la configuración de userdata. Para obtener más información, consulte Configurar ajustes de Build, Flash y Test.

Depuración del error “Fallo de la prueba: se esperaban __ resultados, pero se recibieron___”

Es posible que vea errores que apuntan a fallos durante las pruebas. La prueba espera una cantidad determinada de resultados y no se ven durante la prueba. Algunas pruebas de FreeRTOS se ejecutan antes de que IDT vea el resultado del dispositivo. Si aparece este error, puede intentar aumentar el valor de `testStartDelayms` en la configuración de userdata. Para obtener más información, consulte Configurar ajustes de Build, Flash y Test.

Depuración del error “_____ no se ha seleccionado debido a las restricciones de ConditionalTests”

Esto significa que está realizando una prueba en un grupo de dispositivos que no es compatible con la prueba. Esto puede ocurrir con las pruebas E2E OTA. Por ejemplo, al ejecutar el grupo de pruebas `OTADataplaneMQTT` y en el archivo de configuración `device.json`, ha elegido OTA como No o `OTADataPlaneProtocol` como HTTP. El grupo de pruebas elegido para ejecutarse debe coincidir con sus selecciones de capacidad de `device.json`.

Depuración de un tiempo de espera de IDT durante la supervisión de la salida del dispositivo

Se puede agotar el tiempo de espera de IDT debido a varios motivos. Si se agota el tiempo de espera durante la fase de monitorización de la salida del dispositivo de una prueba y puede ver los resultados en el registro de casos de las pruebas de IDT, significa que IDT los analizó incorrectamente. Una de las razones podrían ser los mensajes de registro intercalados en medio de los resultados de las pruebas. Si este es el caso, consulte la Guía de portabilidad de FreeRTOS para obtener más detalles sobre cómo se deben configurar los registros de UNITY.

Otro motivo por el que se agota el tiempo de espera durante la monitorización de la salida del dispositivo podría ser que el dispositivo se reinicie tras fallar una sola prueba de TLS. A continuación, el dispositivo ejecuta la imagen instalada y provoca un bucle infinito que se ve en los registros. Si esto ocurre, asegúrate de que el dispositivo no se reinicie después de un error en la prueba.

Error que indica que “no está autorizado para acceder al recurso”

Es posible que vea el error que indica que el *`usuario/rol`* no está autorizado para acceder a este recurso en la salida del terminal o en el archivo `test_manager.log` de `/results/`*`execution-id`*`/logs`. Para resolver este problema, asocie la política administrada `AWSIoTDeviceTesterForFreeRTOSFullAccess` al usuario de prueba. Para obtener más información, consulte Creación y configuración de una cuenta de AWS.

Depuración de errores de prueba de red

En las pruebas basada en red, IDT inicia un servidor de eco que se vincula a un puerto no reservado en el equipo host. Si experimenta errores relacionados con los tiempos de espera o la disponibilidad de las conexiones en las pruebas de sockets seguros o de Wi-Fi, asegúrese de que la red está configurada para permitir el tráfico a los puertos configurados en el rango 1024-49151.

Las pruebas de sockets seguros utiliza los puertos 33333 y 33334 de forma predeterminada. Las pruebas de Wi-Fi utilizan el puerto 33335 de forma predeterminada. Si estos tres puertos están en uso o bloqueados por el firewall o la red, puede elegir puertos diferentes para las pruebas en userdata.json. Para obtener más información, consulte Configurar ajustes de Build, Flash y Test. Puede utilizar los siguientes comandos para comprobar si un puerto específico está en uso:

- Windows: `netsh advfirewall firewall show rule name=all | grep port`
- Linux: `sudo netstat -pan | grep port`
- macOS: `netstat -nat | grep port`

Errores de actualización de OTA producidos por una carga útil de la misma versión

Si los casos de prueba OTA fallan porque la misma versión está en el dispositivo después de haber realizado una operación OTA, puede deberse a que su sistema de compilación (por ejemplo, cmake) no reconoció los cambios de IDT en el código fuente de FreeRTOS y no creó un binario actualizado. Esto hace que la actualización OTA se realice con el mismo binario que está actualmente en el dispositivo y que la prueba produzca un error. Para solucionar errores de actualización OTA, comience por asegurarse de que está utilizando la última versión compatible de su sistema de compilación.

Fallo de prueba de OTA en caso de prueba de **`PresignedUrlExpired`**

Un requisito previo de esta prueba es que el tiempo de actualización OTA sea superior a 60 segundos; de lo contrario, la prueba producirá un error. Si esto ocurre, encontrará el siguiente mensaje de error en el registro: "Test takes less than 60 seconds (url expired time) to finish. Please reach out to us" (La prueba tarda menos de 60 segundos (tiempo de vencimiento de la url) en finalizar. Póngase en contacto con nosotros.).

Depuración de errores de puerto e interfaz del dispositivo

Esta sección contiene información acerca de las interfaces de dispositivo que IDT utiliza para conectarse a sus dispositivos.

Plataformas admitidas

IDT es compatible con Linux, macOS y Windows. Las tres plataformas tienen diferentes esquemas de nomenclatura para dispositivos de serie que se asocian a ellas:

- Linux: `/dev/tty*`
- macOS: `/dev/tty.*` o `/dev/cu.*`
- Windows: COM*

Para comprobar el puerto del dispositivo:

- En Linux/macOS, abra un terminal y ejecute `ls /dev/tty*`.
- En macOS, abra un terminal y ejecute `ls /dev/tty.*` o `ls /dev/cu.*`.
- En el caso de Windows, abra el Administrador de dispositivos y expanda el grupo de dispositivos de serie.

Para verificar qué dispositivo está conectado a un puerto:

- En Linux, asegúrese de que el paquete `udev` está instalado y, a continuación, ejecute `udevadm info –name=PORT`. Esta utilidad imprime información del controlador de dispositivo que le ayuda a verificar que está utilizando el puerto correcto.
- En macOS, abra Launchpad y busque **`System Information`**.
- En el caso de Windows, abra el Administrador de dispositivos y expanda el grupo de dispositivos de serie.

Interfaces de dispositivos

Cada dispositivo integrado es diferente, lo que significa que pueden tener uno o más puertos de serie. Es frecuente que los dispositivos tengan dos puertos cuando se conectan a un equipo.

- Un puerto de datos para actualizar el dispositivo.
- Un puerto de lectura para leer la salida.

  Debe establecer el puerto de lectura correcto en el archivo `device.json`. De lo contrario, podría no ser posible leer la salida del dispositivo.

  En el caso de varios puertos, asegúrese de utilizar el puerto de lectura del dispositivo en el archivo `device.json`. Por ejemplo, si conecta un dispositivo Espressif WRover y los dos puertos asignados son `/dev/ttyUSB0` y`/dev/ttyUSB1`, use `/dev/ttyUSB1` en el archivo `device.json`.

  En el caso de Windows, siga la misma lógica.

Lectura de datos de dispositivo

IDT para FreeRTOS utiliza herramientas individuales de compilación e instalación de dispositivos para especificar la configuración de los puertos. Si va a probar el dispositivo y no obtiene salida, pruebe los valores predeterminados siguientes:

- Velocidad en baudios: 115 200
- Bits de datos: 8
- Paridad: ninguna
- Bits de parada: 1

- Control del flujo: ninguno

IDT para FreeRTOS administra esta configuración. No tiene que definirla. Sin embargo, puede utilizar el mismo método para leer manualmente la salida del dispositivo. En Linux o macOS, puede hacerlo con el comando `screen`. En Windows, puede utilizar un programa como TeraTerm.

```
Screen: screen /dev/cu.usbserial 115200
```

```
TeraTerm: Use the above-provided settings to set the fields explicitly in
the GUI.
```

Problemas de la cadena de herramientas de desarrollo

En esta sección se explican los problemas que pueden ocurrir con la cadena de herramientas.

Code Composer Studio en Ubuntu

Las versiones más recientes de Ubuntu (17.10 y 18.04) tienen una versión del paquete `glibc` que no es compatible con las versiones de Code Composer Studio 7.x. Se recomienda instalar la versión de Code Composer Studio 8.2 o posterior.

Los síntomas de incompatibilidad podrían incluir:

- FreeRTOS no puede compilarse ni instalarse en el dispositivo.
- El instalador de Code Composer Studio puede bloquearse.
- No se muestra la salida de registro en la consola durante el proceso de compilación o actualización.
- El comando de compilación intenta lanzarse en modo GUI incluso cuando se invoca como sin encabezado.

Registro

Los registros de IDT para FreeRTOS se colocan en una única ubicación. En el directorio IDT raíz, estos archivos están disponibles en `results/`*`execution-id`*`/`:

- `FRQ_Report.xml`
- `awsiotdevicetester_report.xml`
- `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`

`FRQ_Report.xml` y `logs/test_group_id__test_case_id.log` son los registros más importantes que debe examinar. `FRQ_Report.xml` contiene información sobre qué casos de prueba registraron errores con un mensaje específico. Puede utilizar `logs/test_group_id__test_case_id.log` para investigar más a fondo el problema y tener más contexto.

Errores de la consola

Cuando se ejecuta AWS IoT Device Tester, los errores se notifican en la consola con mensajes breves. Busque en `results/execution-id/logs/test_group_id__test_case_id.log` para obtener más información sobre el error.

Errores de registro

Cada ejecución del conjunto de pruebas tiene un ID único que se utiliza para crear una carpeta llamada `results/execution-id`. Los registros de casos de prueba individuales se encuentran en el directorio `results/execution-id/logs`. Utilice la salida de la consola de IDT para FreeRTOS para encontrar el ID de ejecución, el ID del caso de prueba y el ID del grupo de pruebas que tiene errores. A continuación, utilice esta información para buscar y abrir el archivo de registro del caso de prueba llamado `results/execution-id/logs/test_group_id__test_case_id.log`. La información de este archivo contiene la salida completa del comando de compilación e instalación, la salida de la ejecución de la prueba y la salida de la consola de AWS IoT Device Tester más detallada.

Problemas con los buckets de S3

Si pulsa CTRL+C al ejecutar IDT, IDT iniciará un proceso de limpieza. Parte de esa limpieza consiste en eliminar los recursos de Amazon S3 que se crearon como parte de las pruebas de IDT. Si la limpieza no puede finalizar, es posible que se produzca un problema por el hecho de que se hayan creado demasiados buckets de Amazon S3. Esto significa que la próxima vez que ejecute IDT, las pruebas comenzarán a fallar.

Si pulsa CTRL+C para detener IDT, debe dejar que finalice el proceso de limpieza para evitar este problema. También puede eliminar los buckets de Amazon S3 de su cuenta que se crearon manualmente.

Resolución de problemas de errores de tiempo de espera

Si ve errores de tiempo de espera mientras ejecuta un conjunto de pruebas, aumente el tiempo de espera especificando un factor multiplicador de tiempo de espera. Este factor se aplica al valor de

tiempo de espera predeterminado. 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 el conjunto de pruebas.

Example

IDT v3.0.0 and later

```
./devicetester_linux run-suite --suite-id FRQ_1.0.1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

IDT v1.7.0 and earlier

```
./devicetester_linux run-suite --suite-id FRQ_1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

Característica móvil y cargos de AWS

Si la característica `Cellular` está establecida en `Yes` en su archivo `device.JSON`, FullSecureSockets utilizará instancias de EC2 t2.micro para ejecutar las pruebas y esto puede incurrir en costes adicionales en su cuenta de AWS. Para obtener más información, consulte Precios de Amazon EC2.

Política de generación de informes de calificación

Los informes de calificación solo los generan las versiones de AWS IoT Device Tester (IDT) compatibles con las versiones de FreeRTOS publicadas en los últimos dos años. Si tiene alguna pregunta acerca de la política de compatibilidad, póngase en contacto con AWS Support.

Política administrada de AWS para AWS IoT Device Tester

Una política administrada de AWS es una política independiente que AWS crea y administra. Las políticas administradas de AWS se diseñan para ofrecer permisos para muchos casos de uso comunes, por lo que puede empezar a asignar permisos a los usuarios, grupos y roles.

Tenga presente que es posible que las políticas administradas de AWS no concedan permisos de privilegio mínimo para los casos de uso concretos, ya que están disponibles para que las utilicen

todos los clientes de AWS. Se recomienda definir políticas administradas por el cliente para los casos de uso a fin de reducir aún más los permisos.

No puede cambiar los permisos definidos en las políticas administradas por AWS. Si AWS actualiza los permisos definidos en un política administrada de AWS, la actualización afecta a todas las identidades de entidades principales (usuarios, grupos y roles) a las que está adjunta la política. Lo más probable es que AWS actualice una política administrada de AWS cuando se lance un nuevo Servicio de AWS o las operaciones de la API nuevas estén disponibles para los servicios existentes.

Para obtener más información, consulte Políticas administradas de AWS en la Guía del usuario de IAM.

Temas

- Política administrada de AWS: AWSIoTDeviceTesterForFreeRTOSFullAccess
- Actualizaciones de AWS IoT Device Tester en las políticas administradas de AWS

Política administrada de AWS: AWSIoTDeviceTesterForFreeRTOSFullAccess

La política administrada `AWSIoTDeviceTesterForFreeRTOSFullAccess` contiene los siguientes permisos de AWS IoT Device Tester para la verificación de versiones, las características de actualización automática y la recopilación de métricas.

Detalles del permiso

Esta política incluye los siguientes permisos:

- `iot-device-tester:SupportedVersion`

  Otorga a AWS IoT Device Tester el permiso para obtener la lista de productos, conjuntos de pruebas y versiones de IDT compatibles.

- `iot-device-tester:LatestIdt`

  Otorga a AWS IoT Device Tester el permiso para obtener la versión de IDT más reciente que se puede descargar.

- `iot-device-tester:CheckVersion`

Otorga a AWS IoT Device Tester el permiso para comprobar la compatibilidad de las versiones de IDT, los conjuntos de pruebas y los productos.

- `iot-device-tester:DownloadTestSuite`

  Otorga a AWS IoT Device Tester el permiso para descargar conjuntos de pruebas.

- `iot-device-tester:SendMetrics`

  Otorga a AWS el permiso para recopilar métricas sobre el uso interno de AWS IoT Device Tester.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt-*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iot.amazonaws.com"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteThing",
                "iot:AttachThingPrincipal",
                "iot:DeleteCertificate",
                "iot:GetRegistrationCode",
                "iot:CreatePolicy",
                "iot:UpdateCACertificate",
                "s3:ListBucket",
                "iot:DescribeEndpoint",
                "iot:CreateOTAUpdate",
                "iot:CreateStream",
                "signer:ListSigningJobs",
                "acm:ListCertificates",
                "iot:CreateKeysAndCertificate",
```

```
                "iot:UpdateCertificate",
                "iot:CreateCertificateFromCsr",
                "iot:DetachThingPrincipal",
                "iot:RegisterCACertificate",
                "iot:CreateThing",
                "iam:ListRoles",
                "iot:RegisterCertificate",
                "iot:DeleteCACertificate",
                "signer:PutSigningProfile",
                "s3:ListAllMyBuckets",
                "signer:ListSigningPlatforms",
                "iot-device-tester:SendMetrics",
                "iot-device-tester:SupportedVersion",
                "iot-device-tester:LatestIdt",
                "iot-device-tester:CheckVersion",
                "iot-device-tester:DownloadTestSuite"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "signer:StartSigningJob",
                "acm:GetCertificate",
                "signer:DescribeSigningJob",
                "s3:CreateBucket",
                "execute-api:Invoke",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "signer:CancelSigningProfile"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:098862408343:9xpmnvs5h4/prod/POST/
metrics",
                "arn:aws:signer:*:*:/signing-profiles/*",
                "arn:aws:signer:*:*:/signing-jobs/*",
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:acm:*:*:certificate/*",
                "arn:aws:s3:::idt-*",
                "arn:aws:s3:::afr-ota*"
            ]
        },
```

```
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteStream",
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "iot:DeletePolicy",
                "s3:ListBucketVersions",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "iot:DeleteOTAUpdate",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*",
                "arn:aws:iot:*:*:thinggroup/idt*",
                "arn:aws:iam::*:role/idt-*"
            ]
        },
        {
            "Sid": "VisualEditor4",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "s3:DeleteObjectVersion",
                "iot:DeleteOTAUpdate",
                "s3:PutObject",
                "s3:GetObject",
                "iot:DeleteStream",
                "iot:DeletePolicy",
                "s3:DeleteObject",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "s3:GetObjectVersion",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*",
                "arn:aws:iot:*:*:policy/idt*",
```

```
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iot:*:*:otaupdate/idt*",
                "arn:aws:iot:*:*:thing/idt*",
                "arn:aws:iot:*:*:cert/*",
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:stream/*"
            ]
        },
        {
            "Sid": "VisualEditor5",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*"
            ]
        },
        {
            "Sid": "VisualEditor6",
            "Effect": "Allow",
            "Action": [
                "iot:CancelJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:thing/idt*"
            ]
        },
        {
            "Sid": "VisualEditor7",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
```

```
            }
        },
        {
            "Sid": "VisualEditor8",
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:DeleteSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor9",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor10",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:image/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:key-pair/*",
```

```
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:snapshot/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:subnet/*"
            ]
        },
        {
            "Sid": "VisualEditor11",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor12",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ssm:DescribeParameters",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor13",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
```

```
                    "aws:TagKeys": [
                        "Owner"
                    ]
                },
                "StringEquals": {
                    "ec2:CreateAction": [
                        "RunInstances",
                        "CreateSecurityGroup"
                    ]
                }
            }
        }
    ]
}
```

Actualizaciones de AWS IoT Device Tester en las políticas administradas de AWS

Puede consultar los detalles sobre las actualizaciones de las políticas administradas de AWS para AWS IoT Device Tester desde el momento en que este servicio empezó a realizar el seguimiento de estos cambios.

Versión	Cambio	Descripción	Fecha
7 (más reciente)	Se han reestructurado las condiciones de `ec2:CreateTags` .	Se ha eliminado el uso de `ForAnyValues` .	14/6/2023
6	Se ha eliminado `freertos:ListHardwarePlatforms` de la política.	Se eliminan los permisos, ya que esta acción está en desuso desde el 1 de marzo de 2023.	2/6/2023
5	Se han añadido permisos para ejecutar pruebas del servidor Echo mediante EC2.	Se utiliza para iniciar y detener una instancia de EC2 en las cuentas de AWS de los clientes.	15/12/2020

Versión	Cambio	Descripción	Fecha
4	`iot:CancelJobExecution` añadido.	Este permiso cancela los trabajos OTA.	17/7/2020

Versión	Cambio	Descripción	Fecha
3	Se han añadido los siguientes permisos: • `iot-device-tester:DownloadTestSuite` , • `iot-device-tester:CheckVersion` , • `iot-device-tester:LatestIdt` , • `iot-device-tester:SupportedVersion` .	• `iot-device-tester:DownloadTestSuite` - Otorga a AWS IoT Device Tester el permiso para descargar actualizaciones de los conjuntos de pruebas. • `iot-device-tester:CheckVersion` - Otorga a AWS IoT Device Tester el permiso para comprobar la compatibilidad de las versiones para IDT, los conjuntos de pruebas y los productos. • `iot-device-tester:LatestIdt` - Otorga a AWS IoT Device Tester el permiso para obtener la versión de IDT más reciente que se puede descargar.	23/3/2020

Versión	Cambio	Descripción	Fecha
		• `iot-device-tester: SupportedVersion` - Otorga a AWS IoT Device Tester el permiso para obtener la lista de productos, conjuntos de pruebas y versiones de IDT compatibles.	
2	Se han añadido permisos de `iot-device-tester: SendMetrics`	Otorga a AWS el permiso para recopilar métricas sobre el uso interno de AWS IoT Device Tester.	18/2/2020
1	Versión inicial.		12/2/2020

Política de compatibilidad para AWS IoT Device Tester para FreeRTOS

Important

A partir de octubre de 2022, AWS IoT Device Tester para la calificación de AWS IoT para FreeRTOS (FRQ) 1.0 no genera informes de calificación firmados. No puede calificar nuevos dispositivos de FreeRTOS de AWS IoT para incluirlos en el Catálogo de dispositivos de socios de AWS a través del Programa de calificación de dispositivos de AWS con las versiones IDT FRQ 1.0. Si bien no puede calificar los dispositivos FreeRTOS con IDT FRQ 1.0, puede seguir probando los dispositivos FreeRTOS con FRQ 1.0. Le recomendamos que utilice IDT FRQ 2.0 para calificar y enumerar los dispositivos FreeRTOS en el Catálogo de dispositivos de socios de AWS.

AWS IoT Device Tester para FreeRTOS es una herramienta de automatización de pruebas para validar la portabilidad de FreeRTOS a los dispositivos. Además, puede calificar sus dispositivos FreeRTOS e incluirlos en el Catálogo de dispositivos de socios de AWS. AWS IoT Device Tester para FreeRTOS admite la validación y calificación de las bibliotecas de soporte a largo plazo (LTS) de FreeRTOS disponibles en GitHub, en FreeRTOS/FreeRTOS-LTS y de la línea principal de FreeRTOS, disponible en FreeRTOS/FreeRTOS. Le recomendamos que utilice las versiones más recientes de FreeRTOS y AWS IoT Device Tester para FreeRTOS para validar y calificar los dispositivos.

Para FreeRTOS-LTS, IDT admite la validación y calificación de la versión de FreeRTOS 202210 LTS. Consulte aquí para obtener más información sobre las versiones de FreeRTOS LTS y sus plazos de mantenimiento. Una vez que finalice el período de soporte de estas versiones LTS, podrá continuar con la validación, pero IDT no generará un informe que le permita enviar su dispositivo para la calificación.

Para la línea principal de FreeRTOS, disponible en FreeRTOS/FreeRTOS , se admite la validación y calificación de todas las versiones publicadas en los últimos seis meses o las dos versiones anteriores de FreeRTOS si se publicaron con más de seis meses de diferencia. Consulte aquí las versiones admitidas actualmente. Para las versiones sin compatibilidad de FreeRTOS podrá continuar con la validación, pero IDT no generará un informe que le permita enviar su dispositivo para la calificación.

Consulte Versiones compatibles de AWS IoT Device Tester para FreeRTOS para conocer las últimas versiones compatibles de IDT y FreeRTOS. Puede utilizar cualquiera de las versiones compatibles de AWS IoT Device Tester con la versión correspondiente de FreeRTOS para probar o calificar los dispositivos. Si sigue utilizando las Versiones de IDT no compatibles para FreeRTOS, no recibirá las actualizaciones ni las correcciones de errores más recientes.

Si tiene alguna pregunta acerca de la política de compatibilidad, póngase en contacto con el Servicio de atención al cliente de AWS.

Seguridad en AWS

La seguridad en la nube de AWS es la máxima prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos diseñados para satisfacer los requisitos de seguridad de las organizaciones más exigentes.

La seguridad es una responsabilidad compartida entre AWS y el usuario. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta los servicios de AWS en la nube de AWS AWS también proporciona servicios que puede utilizar de forma segura. Auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad en el marco de los programas de conformidad de AWS. Para obtener más información acerca de los programas de conformidad que se aplican a un servicio de AWS, 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. Usted también es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Esta documentación lo ayudará a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza AWS. Los siguientes temas le mostrarán cómo configurar AWS para satisfacer sus objetivos de seguridad y de conformidad. También obtendrá información sobre cómo utilizar servicios de AWS que pueden ayudarle a supervisar y proteger sus recursos de AWS.

Para obtener información más detallada sobre la seguridad de AWS IoT, consulte Seguridad e identidad para AWS IoT.

Temas

Identity and Access Management para FreeRTOS

AWS Identity and Access Management (IAM) es un Servicio de AWS que ayuda a los administradores a controlar de forma segura el acceso a los recursos de AWS. Los administradores de IAM controlan quién puede estar autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar recursos de FreeRTOS. IAM es un servicio de Servicio de AWS que se puede utilizar sin cargo adicional.

Temas

Público

La forma en que utilice AWS Identity and Access Management (IAM) difiere en función del trabajo que realice en FreeRTOS.

Usuario de servicio - Si utiliza el servicio de FreeRTOS para realizar su trabajo, su administrador le proporcionará las credenciales y los permisos que necesita. A medida que utilice más características de FreeRTOS 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 FreeRTOS, consulte Solución de problemas de identidades y accesos en FreeRTOS.

Administrador de servicio - Si está a cargo de los recursos de FreeRTOS en su empresa, es probable que tenga acceso completo a FreeRTOS. Su trabajo consiste en determinar a qué características y recursos de FreeRTOS deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo su empresa puede utilizar IAM con FreeRTOS, consulte Cómo funciona FreeRTOS con IAM.

Administrador de IAM - Si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a FreeRTOS. Para consultar ejemplos de políticas basadas en la identidad de FreeRTOS que puede utilizar en IAM, consulte Ejemplos de políticas basadas en identidades de FreeRTOS.

Autenticación con identidades

La autenticación es la manera de iniciar sesión en AWS mediante credenciales de identidad. Debe estar autenticado (haber iniciado sesión en AWS) como Usuario raíz de la cuenta de AWS, como un usuario de IAM o asumiendo un rol de IAM.

Puede iniciar sesión en AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad de AWS IAM Identity Center. Los usuarios (del IAM Identity Center), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su administrador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accede a AWS mediante la federación, está asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en la AWS Management Console o en el portal de acceso a AWS. Para obtener más información sobre el inicio de sesión en AWS, consulte Cómo iniciar sesión en su Cuenta de AWS en la Guía del usuario de AWS Sign-In.

Si accede a AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de la línea de comandos (CLI) para firmar criptográficamente las solicitudes mediante el uso de las credenciales. Si no usa las herramientas de AWS, debe firmar usted mismo las solicitudes. Para obtener más información sobre el método recomendado para la firma de solicitudes, consulte Firma de solicitudes API de AWS en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, AWS le recomienda el uso de la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactor en la Guía del usuario de AWS IAM Identity Center y Uso de la autenticación multifactor (MFA) en AWS en la Guía del usuario de IAM.

Usuario raíz de Cuenta de AWS

Cuando se crea una Cuenta de AWS, se comienza con una identidad de inicio de sesión que tiene acceso completo a todos los recursos y Servicios de AWS de la cuenta. Esta identidad recibe el nombre de usuario raíz de la Cuenta de AWS y se accede a ella iniciando sesión con el email y la contraseña que utilizó para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren

que inicie sesión como usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, solicite que los usuarios humanos, incluidos los que requieren acceso de administrador, utilicen la federación con un proveedor de identidades para acceder a Servicios de AWS utilizando credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidad web, el AWS Directory Service, el directorio del Identity Center, o cualquier usuario que acceda a Servicios de AWS utilizando credenciales proporcionadas a través de una fuente de identidad. Cuando identidades federadas acceden a las Cuentas de AWS, asumen roles y los roles proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utilice AWS IAM Identity Center. Puede crear usuarios y grupos en el IAM Identity Center o puede conectarse y sincronizar con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todas sus aplicaciones y Cuentas de AWS. Para obtener más información, consulte ¿Qué es el IAM Identity Center? en la Guía del usuario de AWS IAM Identity Center.

Usuarios y grupos de IAM

Un usuario de IAM es una identidad en su Cuenta de AWS que dispone de permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración en la Guía del Usuario de IAM.

Un grupo de IAM es una identidad que especifica un conjunto de usuarios de IAM. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales.

Para más información, consulte Cuándo crear un usuario de IAM (en lugar de un rol) en la Guía del Usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad de tu Cuenta de AWS que dispone de permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una determinada persona. Puede asumir temporalmente un rol de IAM en la AWS Management Consolecambiando de roles. Puede asumir un rol llamando a una operación de AWS CLI o de la API de AWS, o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulte Uso de roles de IAM en la Guía del Usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles para federación, consulte Creación de un rol para un proveedor de identidades de terceros en la Guía del Usuario de IAM. Si utiliza el IAM Identity Center, debe configurar un conjunto de permisos. El IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulte Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center.
- Permisos de usuario de IAM temporales: un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- Acceso entre cuentas: puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. No obstante, con algunos Servicios de AWS se puede asociar una política directamente a un recurso (en lugar de utilizar un rol como representante). Para obtener información sobre la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.
- Acceso entre servicios: algunos Servicios de AWS utilizan características de otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado a servicios.

- Reenviar sesiones de acceso (FAS): cuando utiliza un rol o un usuario de IAM para llevar a cabo acciones en AWS, se le considera una entidad principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos de la entidad principal para llamar a un Servicio de AWS, combinados con el Servicio de AWS solicitante para realizar solicitudes a servicios posteriores. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS o recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.
- Rol de servicio: un rol de servicio es un rol de IAM que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.
- Rol vinculado a servicios: un rol vinculado a servicios es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados a servicios aparecen en la Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

- Aplicaciones que se ejecutan en Amazon EC2: puede utilizar un rol de IAM que le permita administrar credenciales temporales para las aplicaciones que se ejecutan en una instancia de EC2 y realizan solicitudes a la AWS CLI o a la API de AWS. Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia EC2. Para asignar un rol de AWS a una instancia de EC2 y ponerla a disposición de todas las aplicaciones, cree un perfil de instancia asociado a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia EC2 obtener credenciales temporales. Para obtener más información, consulte Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias de Amazon EC2 en la Guía del usuario de IAM.

Para obtener información sobre el uso de los roles de IAM, consulte Cuándo crear un rol de IAM (en lugar de un usuario) en la Guía del Usuario de IAM.

Administración de acceso mediante políticas

Para controlar el acceso en AWS, se crean políticas y se adjuntan a identidades o recursos de AWS. Una política es un objeto de AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando una entidad principal (sesión de rol, usuario o usuario

raíz) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan en AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte Información general de las políticas JSON en la Guía del Usuario de IAM.

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Para conceder permiso a los usuarios para realizar acciones en los recursos que necesiten, un administrador de IAM puede crear políticas de IAM. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con dicha política puede obtener información del rol de la AWS Management Console, la AWS CLI o la API de AWS.

Políticas basadas en identidades

Las políticas basadas en identidades son documentos de políticas de permisos JSON que puede adjuntar a una identidad, como un usuario, un grupo de usuarios o un rol de IAM. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede asociar a varios usuarios, grupos y roles de su Cuenta de AWS. Las políticas administradas incluyen las políticas administradas de AWS y las políticas administradas por el cliente. Para obtener más información sobre cómo elegir una política administrada o una política insertada, consulte Elegir entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM 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. Las entidades 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 se puede utilizar políticas de IAM administradas de AWS en una política basada en recursos.

Listas de control de acceso (ACL)

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3, AWS WAF y Amazon VPC son ejemplos de servicios que admiten las ACL. Para obtener más información sobre las ACL, consulte Información general de Lista de control de acceso (ACL) en la Guía para Desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite otros tipos de políticas adicionales menos frecuentes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que 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 Límites de permisos para las entidades de IAM en la Guía del Usuario de IAM.
- Políticas de control de servicio (SCP): las SCP son políticas de JSON que especifican los permisos máximos de una organización o una unidad organizativa en AWS Organizations. AWS Organizations es un servicio que le permite agrupar y administrar de manera centralizada varias Cuentas de AWS que posea su empresa. Si habilita todas las características en una empresa, entonces podrá aplicar políticas de control de servicio (SCP) a una o todas sus cuentas. Una SCP limita los permisos para las entidades de las cuentas de miembros, incluido cada Usuario raíz de

la cuenta de AWS. Para obtener más información acerca de Organizations y las SCP, consulte Funcionamiento de las SCP en la Guía del usuario de AWS Organizations.

- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte Políticas de sesión en la Guía del Usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para obtener información sobre cómo AWS decide si permite o no una solicitud cuando hay varios tipos de políticas implicados, consulte Lógica de evaluación de políticas en la Guía del usuario de IAM.

Cómo funciona FreeRTOS con IAM

Antes de utilizar IAM para administrar el acceso a FreeRTOS, conozca qué características de IAM se pueden utilizar con FreeRTOS.

Características de IAM que puede utilizar con FreeRTOS

Característica de IAM	Compatibilidad con FreeRTOS
Políticas basadas en identidades	Sí
Políticas basadas en recursos	No
Acciones de políticas	Sí
Recursos de políticas	Sí
Claves de condición de política (específicas del servicio)	Sí
ACL	No

Característica de IAM	Compatibilidad con FreeRTOS
ABAC (etiquetas en políticas)	Parcial
Credenciales temporales	Sí
Permisos de entidades principales	Sí
Roles de servicio	Sí
Roles vinculados al servicio	No

Para obtener información general sobre cómo funcionan FreeRTOS y otros servicios de AWS con la mayoría de las características de IAM, consulte Servicios de AWS que funcionan con IAM en la Guía del usuario de IAM.

Políticas basadas en identidades para FreeRTOS

Compatibilidad con las políticas basadas en identidades	Sí

Las políticas basadas en identidades son documentos de políticas de permisos JSON que puede adjuntar a una identidad, como un usuario, un grupo de usuarios o un rol de IAM. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Con las políticas basadas en identidades de IAM, puede especificar las acciones y los recursos permitidos o denegados, así como las condiciones en las que se permiten o deniegan las acciones. No es posible especificar la entidad principal en una política basada en identidad porque se aplica al usuario o rol al que está adjunto. Para obtener más información sobre los elementos que puede utilizar en una política JSON, consulte Referencia de los elementos de las políticas JSON de IAM en la Guía del usuario de IAM.

Ejemplos de políticas basadas en identidades de FreeRTOS

Para ver ejemplos de políticas basadas en identidades de FreeRTOS, consulte Ejemplos de políticas basadas en identidades de FreeRTOS.

Políticas basadas en recursos en FreeRTOS

Compatibilidad con las políticas basadas en recursos	No

Las políticas basadas en recursos son documentos de políticas JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM 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. Las entidades principales pueden incluir cuentas, usuarios, roles, usuarios federados o servicios de Servicios de AWS.

Para habilitar el acceso entre cuentas, puede especificar toda una cuenta o entidades de IAM de otra cuenta como la entidad principal de una política en función de recursos. Añadir a una política en función de recursos una entidad principal entre cuentas es solo una parte del establecimiento de una relación de confianza. Cuando la entidad principal y el recurso se encuentran en Cuentas de AWS diferentes, un administrador de IAM de la cuenta de confianza también debe conceder a la entidad principal (usuario o rol) permiso para acceder al recurso. Para conceder el permiso, adjunte la entidad a una política basada en identidad. Sin embargo, si la política en función de recursos concede el acceso a una entidad principal de la misma cuenta, no es necesaria una política basada en identidad adicional. Para más información, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del Usuario de IAM.

Acciones de políticas para FreeRTOS

Admite acciones de políticas	Sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para permitir o denegar el acceso en una política. Las acciones de la política generalmente tienen el mismo nombre que la operación de API de AWS asociada. Hay algunas excepciones, como acciones de solo permiso que no tienen una operación de API coincidente. También hay algunas operaciones que requieren varias acciones en una política. Estas acciones adicionales se denominan acciones dependientes.

Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Para ver una lista de las acciones de FreeRTOS, consulte Acciones definidas por FreeRTOS en la Referencia de autorizaciones de servicio.

Las acciones de políticas de FreeRTOS utilizan el siguiente prefijo antes de la acción:

```
awes
```

Para especificar varias acciones en una única instrucción, sepárelas con comas.

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

Para ver ejemplos de políticas basadas en identidades de FreeRTOS, consulte Ejemplos de políticas basadas en identidades de FreeRTOS.

Recursos de políticas para FreeRTOS

Admite recursos de políticas	Sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). Puede

hacerlo para acciones que admitan un tipo de recurso específico, conocido como permisos de nivel de recurso.

Para las acciones que no admiten permisos de nivel de recurso, como las operaciones de descripción, utilice un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

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

Para ver una lista de los tipos de recursos de FreeRTOS y sus ARN, consulte Recursos definidos por FreeRTOS en la Referencia de autorizaciones de servicio. Para obtener información sobre las acciones con las que puede especificar el ARN de cada recurso, consulte Acciones definidas por FreeRTOS.

Para ver ejemplos de políticas basadas en identidades de FreeRTOS, consulte Ejemplos de políticas basadas en identidades de FreeRTOS.

Claves de condición de política para FreeRTOS

Admite claves de condición de políticas específicas del servicio	Sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` (o bloque de `Condition`) permite especificar condiciones en las que entra en vigor una instrucción. El elemento `Condition` es opcional. Puede crear expresiones condicionales que utilicen operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud.

Si especifica varios elementos de `Condition` en una instrucción o varias claves en un único elemento de `Condition`, AWS las evalúa mediante una operación lógica `AND`. Si especifica varios valores para una única clave de condición, AWS evalúa la condición con una operación `OR` lógica. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puede utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puede conceder un permiso de usuario de IAM para acceder a un recurso solo si está etiquetado

con su nombre de usuario de IAM. Para más información, consulte Elementos de la política de IAM: variables y etiquetas en la Guía del usuario de IAM.

AWS admite claves de condición globales y claves de condición específicas del servicio. Para ver todas las claves de condición globales de AWS, consulte Claves de contexto de condición globales de AWS en la Guía del Usuario de IAM.

Para ver una lista de las claves de condición de FreeRTOS, consulte Claves de condición para FreeRTOS en la Referencia de autorizaciones de servicio. Para obtener más información sobre las acciones y los recursos con los que puede utilizar una clave de condición, consulte Acciones definidas por FreeRTOS.

Para ver ejemplos de políticas basadas en identidades de FreeRTOS, consulte Ejemplos de políticas basadas en identidades de FreeRTOS.

ACL en FreeRTOS

Admite las ACL	No

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

ABAC con FreeRTOS

Admite ABAC (etiquetas en las políticas)	Parcial

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define permisos en función de atributos. En AWS, estos atributos se denominan etiquetas. Puede adjuntar etiquetas a entidades de IAM (usuarios o roles) y a muchos recursos de AWS. El etiquetado de entidades y recursos es el primer paso de ABAC. A continuación, designa las políticas de ABAC para permitir operaciones cuando la etiqueta de la entidad principal coincida con la etiqueta del recurso al que se intenta acceder.

ABAC es útil en entornos que crecen con rapidez y ayuda en situaciones en las que la administración de las políticas resulta engorrosa.

Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Si un servicio admite las tres claves de condición para cada tipo de recurso, el valor es Sí para el servicio. Si un servicio admite las tres claves de condición solo para algunos tipos de recursos, el valor es Parcial.

Para obtener más información sobre ABAC, consulte ¿Qué es ABAC? en la Guía del Usuario de IAM. Para ver un tutorial con los pasos para configurar ABAC, consulte Uso del control de acceso basado en atributos (ABAC) en la Guía del Usuario de IAM.

Uso de credenciales temporales con FreeRTOS

Admite el uso de credenciales temporales	Sí

Algunos Servicios de AWS no funcionan cuando inicia sesión con credenciales temporales. Para obtener información adicional, incluida la información sobre qué Servicios de AWS funcionan con credenciales temporales, consulte Servicios de AWS que funcionan con IAM en la Guía del usuario de IAM.

Utilice credenciales temporales si inicia sesión en la AWS Management Console con cualquier método excepto un nombre de usuario y una contraseña. Por ejemplo, cuando accede a AWS utilizando el enlace de inicio de sesión único (SSO) de la empresa, ese proceso crea automáticamente credenciales temporales. También crea automáticamente credenciales temporales cuando inicia sesión en la consola como usuario y luego cambia de rol. Para obtener más información sobre el cambio de roles, consulte Cambio a un rol (consola) en la Guía del usuario de IAM.

Puede crear credenciales temporales de forma manual mediante la AWS CLI o la API de AWS. A continuación, puede usar esas credenciales temporales para acceder a AWS. AWS recomienda generar credenciales temporales de forma dinámica en lugar de usar claves de acceso a largo plazo. Para más información, consulte Credenciales de seguridad temporales en IAM.

Permisos de entidades principales entre servicios para FreeRTOS

Admite Forward access sessions (FAS)	Sí

Cuando utiliza un usuario o un rol de IAM para llevar a cabo acciones en AWS, se lo considera una entidad principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos de la entidad principal para llamar a un Servicio de AWS, combinados con el Servicio de AWS solicitante para realizar solicitudes a servicios posteriores. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS o recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.

Roles de servicio para FreeRTOS

Compatible con roles de servicio	Sí

Un rol de servicio es un rol de IAM que asume un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.

> **Warning**
>
> Cambiar los permisos de un rol de servicio podría interrumpir la funcionalidad de FreeRTOS. Edite los roles de servicio solo cuando FreeRTOS proporcione orientación para hacerlo.

Roles vinculados a servicios para FreeRTOS

Admite roles vinculados a servicios	No

Un rol vinculado a servicios es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados a servicios aparecen en la Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

Para más información sobre cómo crear o administrar roles vinculados a servicios, consulte Servicios de AWS que funcionan con IAM. Busque un servicio en la tabla que incluya `Yes` en la columna Rol

vinculado a un servicio. Seleccione el vínculo Sí para ver la documentación acerca del rol vinculado a servicios para ese servicio.

Ejemplos de políticas basadas en identidades de FreeRTOS

De forma predeterminada, los usuarios y roles no tienen permiso para crear o modificar recursos de FreeRTOS. Tampoco pueden realizar tareas mediante la AWS Management Console, la AWS Command Line Interface (AWS CLI) o la API de AWS. Para conceder permiso a los usuarios para realizar acciones en los recursos que necesiten, un administrador de IAM puede crear políticas de IAM. A continuación, el administrador puede añadir las políticas de IAM a roles, y los usuarios pueden asumirlos.

Para obtener información sobre cómo crear una política basada en identidad de IAM mediante el uso de estos documentos de políticas JSON de ejemplo, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

A fin de obtener más información sobre las acciones y los tipos de recursos definidos por FreeRTOS, incluido el formato de los ARN para cada tipo de recurso, consulte Acciones, recursos y claves de condición para FreeRTOS en la Referencia de autorizaciones de servicio.

Temas

- Prácticas recomendadas sobre las políticas
- Uso de la consola de FreeRTOS
- Cómo permitir a los usuarios consultar sus propios permisos

Prácticas recomendadas sobre las políticas

Las políticas basadas en identidades determinan si alguien puede crear, acceder o eliminar los recursos de FreeRTOS de la cuenta. Estas acciones pueden generar costes adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience con las políticas administradas de AWS y continúe con los permisos de privilegio mínimo: a fin de comenzar a conceder permisos a los usuarios y las cargas de trabajo, utilice las políticas administradas de AWS, que conceden permisos para muchos casos de uso comunes. Están disponibles en su Cuenta de AWS. Se recomienda definir políticas administradas por el cliente de AWS específicas para los casos de uso a fin de reducir aún más los permisos. Con

el fin de obtener más información, consulte las políticas administradas por AWS o las políticas administradas por AWS para funciones de trabajo en la Guía del usuario de IAM.

- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulte Políticas y permisos en IAM en la Guía de usuario de IAM.
- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puede escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puede usar condiciones para conceder acceso a acciones de servicios si se emplean a través de un Servicio de AWS determinado, como por ejemplo AWS CloudFormation. Para obtener más información, consulte Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.
- Utilice el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. xPara 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 la MFA a sus políticas. Para obtener más información, consulte Configuración de acceso a una API protegida por MFA en la Guía del usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte las Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.

Uso de la consola de FreeRTOS

Para acceder a la consola de FreeRTOS, debe tener un conjunto mínimo de permisos. Estos permisos deben permitirle registrar y consultar los detalles acerca de los recursos de FreeRTOS en su Cuenta de AWS. Si crea una política basada en identidades que sea más restrictiva que el mínimo

de permisos necesarios, la consola no funcionará del modo esperado para las entidades (usuarios o roles) que tengan esa política.

No es necesario que conceda permisos mínimos para la consola a los usuarios que solo realizan llamadas a la AWS CLI o a la API de AWS. En su lugar, permite acceso únicamente a las acciones que coincidan con la operación de API que intentan realizar.

Para asegurarse de que los usuarios y los roles puedan seguir utilizando la consola de FreeRTOS, asocie también a las entidades la política administrada de AWS *ConsoleAccess* o *ReadOnly* de FreeRTOS. Para obtener más información, consulte Adición de permisos a un usuario en la Guía del usuario de IAM.

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas administradas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para realizar esta acción en la consola o mediante programación con la AWS CLI o la API de AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
```

```
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Solución de problemas de identidades y accesos en FreeRTOS

Utilice la siguiente información para diagnosticar y solucionar los problemas comunes que puedan surgir cuando trabaje con FreeRTOS e IAM.

Temas

- No tengo autorización para realizar una acción en FreeRTOS
- No estoy autorizado a realizar actividades como: PassRole
- Quiero permitir a personas externas a mi Cuenta de AWS acceder a mis recursos de FreeRTOS

No tengo autorización para realizar una acción en FreeRTOS

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM `mateojackson` intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio *`my-example-widget`*, pero no tiene los permisos ficticios `awes:`*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

En este caso, la política del usuario `mateojackson` debe actualizarse para permitir el acceso al recurso *`my-example-widget`* mediante la acción `awes:`*`GetWidget`*.

Si necesita ayuda, póngase en contacto con su administrador de AWS. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

No estoy autorizado a realizar actividades como: PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, se deben actualizar las políticas a fin de permitirle transferir un rol a FreeRTOS.

Algunos Servicios de AWS le permiten transferir un rol existente a dicho servicio en lugar de crear un nuevo rol de servicio o uno vinculado al servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en FreeRTOS. Sin embargo, la acción requiere que el servicio cuente con permisos que concede un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su administrador de AWS. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir a personas externas a mi Cuenta de AWS acceder a mis recursos de FreeRTOS

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para más información, consulte lo siguiente:

- Para obtener información acerca de si FreeRTOS admite estas características, consulte Cómo funciona FreeRTOS con IAM.
- Para obtener información acerca de cómo proporcionar acceso a los recursos de las Cuentas de AWS de su propiedad, consulte Proporcionar acceso a un usuario de IAM a otra Cuenta de AWS de la que es propietario en la Guía del usuario de IAM.

- Para obtener información acerca de cómo proporcionar acceso a tus recursos a Cuentas de AWS de terceros, consulte Proporcionar acceso a Cuentas de AWS que son propiedad de terceros en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante federación de identidades, consulte Proporcionar acceso a usuarios autenticados externamente (federación de identidades) en la Guía del usuario de IAM.
- Para obtener información sobre la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del Usuario de IAM.

Validación de conformidad

Para saber si un Servicio de AWS está incluido en el ámbito de programas de conformidad específicos, consulte Servicios de AWS en el ámbito del programa de conformidad y elija el programa de conformidad que le interese. Para obtener información general, consulte Programas de conformidad de AWS.

Puede descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact.

Su responsabilidad de conformidad al utilizar Servicios de AWS se determina en función de la sensibilidad de los datos, los objetivos de conformidad de su empresa y la legislación y los reglamentos correspondientes. AWS proporciona los siguientes recursos para ayudar con la conformidad:

- Guías de inicio rápido de seguridad y conformidad: estas guías de implementación tratan consideraciones sobre arquitectura y ofrecen pasos para implementar los entornos de referencia centrados en la seguridad y la conformidad en AWS.
- Arquitectura para la seguridad y el cumplimiento de la HIPAA en Amazon Web Services: en este documento técnico, se describe cómo las empresas pueden utilizar AWS para crear aplicaciones aptas para HIPAA.

> **Note**
>
> No todos los Servicios de AWS son aptos para HIPAA. Para obtener más información, consulte la Referencia de servicios aptos para HIPAA.

- Recursos de conformidad de AWS: este conjunto de manuales y guías podría aplicarse a su sector y ubicación.
- Guías de cumplimiento para clientes de AWS: comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. Las guías resumen las mejores prácticas para garantizar la seguridad de los Servicios de AWS y orientan los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés), el Consejo de Estándares de Seguridad de la Industria de Tarjetas de Pago (PCI, por sus siglas en inglés) y la Organización Internacional de Normalización (ISO, por sus siglas en inglés)).
- Evaluación de recursos con reglas en la Guía para desarrolladores de AWS Config: el servicio AWS Config evalúa en qué medida las configuraciones de sus recursos cumplen las prácticas internas, las directrices del sector y las normativas.
- AWS Security Hub: este Servicio de AWS proporciona una visión completa de su estado de seguridad en AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su conformidad 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.
- AWS Audit Manager: este servicio de Servicio de AWS le ayuda a auditar continuamente el uso de AWS con el fin de simplificar la forma en que administra el riesgo y la conformidad con las normativas y los estándares del sector.

Resiliencia en AWS

La infraestructura global de AWS se compone de regiones de AWS y zonas de disponibilidad de AWS. Las regiones proporcionan varias zonas de disponibilidad físicamente independientes y aisladas que se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre zonas de disponibilidad sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Para obtener más información sobre las regiones y zonas de disponibilidad de AWS, consulte Infraestructura global de AWS.

Seguridad de la infraestructura en FreeRTOS

Los servicios administrados de AWS están protegidos por los procedimientos de seguridad de red globales de AWS que se describen en el documento técnico Amazon Web Services: información general sobre procesos de seguridad.

Puede utilizar llamadas a la API publicadas en AWS para obtener acceso a los servicios de AWS a través de la red. Los clientes deben ser compatibles con Transport Layer Security (TLS) 1.2 o una versión posterior. Recomendamos TLS 1.3 o una versión posterior. Los clientes también deben ser compatibles con conjuntos de cifrado con confidencialidad directa total (PFS) tales como Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Guía de migración del repositorio Github de Amazon-FreeRTOS

Si tiene un proyecto FreeRTOS existente basado en el repositorio amazon-freertos, ahora obsoleto, siga estos pasos:

1. Manténgase al día con las últimas correcciones de seguridad disponibles públicamente. Consulte la página de Bibliotecas de FreeRTOS LTS para ver las actualizaciones o suscríbase al repositorio de GitHub de FreeRTOS-LTS para recibir los últimos parches de LTS con correcciones de errores críticos y de seguridad. Puede descargar o clonar los últimos parches de FreeRTOS LTS necesarios directamente desde los repositorios individuales de GitHub.
2. Considere la posibilidad de refactorizar la implementación de la interfaz de transporte de red para optimizar su plataforma de hardware. La biblioteca más reciente de coreMQTT no requiere API abstractas, como los sockets seguros y las API de WiFi. Consulte Interfaz de transporte para obtener más información.

Apéndice

En la siguiente tabla se proporcionan recomendaciones para todos los proyectos de demostración, las bibliotecas antiguas y las API abstractas del repositorio Amazon-FreeRTOS.

Bibliotecas y demostraciones migradas

Nombre	Tipo	Recomendaciones	
coreHTTP	demostraciones y biblioteca	Clone o descargue la biblioteca coreHTTP directamente desde el repositorio coreHTTP (submódulo si usa git) en la organización Github de FreeRTOS. Las demostraciones de coreHTTP se encuentran en la distribución principal de FreeRTOS. Para obtener más detalles, consulte la página coreHTTP.	

Nombre	Tipo	Recomendaciones	
coreMQTT	demostraciones y biblioteca	Clone o descargue la biblioteca coreMQTT directamente desde el repositorio coreMQTT (submódulo si usa git) en la organización Github de FreeRTOS. Las demostraciones de coreMQTT se encuentran en la distribución principal de FreeRTOS. Para obtener más detalles, consulte la página coreMQTT.	
coreMQTT-Agent	demostraciones y biblioteca	Clone o descargue la biblioteca coreMQTT-Agent directamente desde el repositorio coreMQTT-Agent (submódulo si usa git) en la organización Github de FreeRTOS. Las demostraciones de coreMQTT-Agent se encuentran en el repositor io coreMQTT-Agent-Demos. Para obtener más detalles, consulte la página coreMQTT-Agent.	
device_de fender_for_aws	demostraciones y biblioteca	La biblioteca AWS IoT Device Defender se encuentra en su repositorio, en la organización de GitHub de AWS. Clónela o descárguela (submódulo si usa git) directamente desde el repositorio de AWS IoT Device Defender. Las demostraciones de Device Defender se encuentran en la distribución principal de FreeRTOSAWS IoT. Para obtener más información, consulte la página de AWS IoT Device Defender.	

Nombre	Tipo	Recomendaciones	
device_shadow_for_aws	demostraciones y biblioteca	La biblioteca de sombra de dispositivo de AWS IoT se encuentra en su repositorio, en la organización de GitHub de AWS. Clónela o descárguela (submódulo si usa git) directamente desde el repositorio de sombra de dispositivo de AWS IoT. Las demostraciones de sombra de dispositivo de se encuentran en la distribución principal de FreeRTOSAWS IoT. Para obtener más información, consulte la página de sombra de dispositivo de AWS IoT.	
jobs_for_aws	demostraciones y biblioteca	La biblioteca de trabajos de AWS IoT se encuentra en su repositorio, en la organización de GitHub de AWS. Clónela o descárguela (submódulo si usa git) directamente desde el repositorio de trabajos de AWS IoT. Las demostraciones de trabajos de AWS IoT se encuentran en la distribución principal de FreeRTOS. Para obtener más detalles, consulte la página de trabajos de AWS IoT.	

Nombre	Tipo	Recomendaciones
OTA	demostraciones y biblioteca	La biblioteca de actualizaciones inalámbricas de AWS IoT se encuentra en su repositorio en la organización de GitHub de AWS. Clónela o descárguela (submódul o si usa git) directamente desde el repositorio de OTA de AWS IoT. Las demostraciones de OTA de AWS IoT se encuentran en la distribución principal de FreeRTOS. Para obtener más detalles, consulte la página de OTA de AWS IoT.
CLI y FreeRTOS_ Plus_CLI	demostraciones y biblioteca	Hay un ejemplo de la CLI que se ejecuta en WinSim. Consulte la página de la interfaz de línea de comandos de FreeRTOS Plus para obtener más información. Las integraciones de referencia de FreeRTOS IoT destacadas en las plataformas NXP i.MX RT1060 y STM32U5 también proporcionan ejemplos de CLI en hardware real.
registro	macro	Algunas de las bibliotecas de FreeRTOS utilizan implement aciones de la macro de registro para plataformas de hardware específicas. Consulte la página de registro para saber cómo implementar la macro de registro. Consulte una de las referencias de IoT destacadas de FreeRTOS para ver un ejemplo que se ejecuta en hardware real.

Nombre	Tipo	Recomendaciones	
greengrass_connectivity	demostración	[Migración en curso] Para este proyecto de demostración se asumió que la conectividad a la nube estaba disponible antes de conectarse a un dispositivo AWS IoT Greengrass. Se está desarrollando un nuevo proyecto que muestra la capacidad local de autenticación y detección. Se espera que el nuevo proyecto de demostración se publique en breve en la organización de Github de FreeRTOS.	

Bibliotecas y demostraciones obsoletas

Nombre	Tipo	Recomendaciones	
BLE	demostraciones y biblioteca	La biblioteca BLE de FreeRTOS implementa el protocolo propietario MQTT y admite la publicación y suscripción a temas MQTT sobre Bluetooth de bajo consumo (BLE) a través de un dispositivo proxy como un teléfono móvil. Esto ya no es obligatorio. Utilice su propia pila de BLE o una opción de terceros, como NimBLE, para optimizar mejor su proyecto.	
dev_mode_key_provisioning	demostraciones	Las integraciones de referencia de IoT de FreeRTOS destacadas en las plataformas NXP i.MX RT1060, STM32U5 o ESP32-C3 proporcio	

Nombre	Tipo	Recomendaciones
		nan ejemplos de aprovisionamiento crucial mediante una CLI.
posix	abstracción y demostración	No se recomienda su uso.
wifi_provisioning	ejemplo	En este ejemplo se muestra cómo aprovisionar credenciales WiFi en un dispositivo mediante la bibliotec a BLE de Amazon FreeRTOS. Consulte la referencia de IoT destacada de FreeRTOS en la plataforma ESP32C3 para ver un ejemplo de aprovisionamiento de WiFi mediante BLE.
API abstractas heredadas	code	Se trata de API que se crearon para proporcionar una interfaz abstracta para varios paquetes de software, módulos de conectividad y plataformas MCU de terceros de diversos proveedores. Por ejemplo, hay interfaces para la abstracci ón de WiFi, sockets seguros, etc. Se admiten en el repositorio de Amazon-FreeRTOS y se encuentra n en la carpeta `/libraries/ abstractions/` . Estas API no son necesarias cuando se utilizan las bibliotecas LTS de FreeRTOS.

Las bibliotecas y demostraciones de la tabla anterior no recibirán parches de seguridad ni correcciones de errores.

Bibliotecas de terceros

Cuando las demostraciones de Amazon-FreeRTOS utilicen bibliotecas de terceros, le recomendamos que las submodule directamente desde sus repositorios de terceros.

- cMock: clónela (submódulo si usa git) directamente desde el repositorio de Cmock.
- jsmn: no se recomienda y ya no se admite.
- lwip: clónela (submódulo si usa git) directamente desde el repositorio de lwip-tcpip.
- lwip_osal: consulte las integraciones de referencia destacadas de FreeRTOS en i.MX RT1060 o STM32U5 para saber cómo implementar lwip_osal en su plataforma o placa de hardware.
- mbedtls: clónela (submódulo si usa git) directamente desde el repositorio de Mbed-TLS. La configuración y las utilidades de mbedtls se pueden reutilizar; en este caso, haga una copia local.
- pkcs11: clónela (submódulo si usa git) directamente desde la biblioteca corePKCS11 o desde el repositorio PKCS 11 de OASIS.
- tinycbor: clónela (submódulo si usa git) directamente desde el repositorio de tinycbor.
- tinycrypt: le recomendamos que utilice aceleradores criptográficos de su plataforma MCU, si están disponibles. Si desea seguir usando tinycrypt, clónela (submódulo si usa git) directamente desde el repositorio de tinycrypt.
- tracealyzer_recorder: clónela (submódulo si usa git) directamente desde el repositorio de grabadora de seguimientos de Percepio.
- unity: clónwlo (submódulo si usa git) directamente desde el repositorio ThrowTheSwitch/Unity.
- win_pcap: win_pcap ya no se mantiene. Le recomendamos que utilice libslirp, libpcap (posix) o npcap en su lugar.

Pruebas de portabilidad y pruebas de integración

Todas las pruebas de la carpeta `/tests` necesarias para validar la integración de las bibliotecas FreeRTOS se migraron al repositorio FreeRTOS-Libraries-Integration-Tests. Se pueden usar para probar la implementación de PAL y la integración de la biblioteca. AWS IoT Device Tester utiliza las mismas pruebas para el Programa de calificación de dispositivos de AWS para FreeRTOS.

Documentación archivada de FreeRTOS

Archivo de guías del usuario de FreeRTOS

Estas versiones anteriores de la Guía del usuario de FreeRTOS están disponibles para su uso con las versiones LTS (soporte a largo plazo) de FreeRTOS.

- Guía del usuario de FreeRTOS para FreeRTOS versión 202210.00
- Guía del usuario de FreeRTOS para FreeRTOS versión 202012.00

Contenido anterior de la Guía del usuario de FreeRTOS

Este contenido está obsoleto, pero se proporciona aquí como referencia.

Consulte Introducción a FreeRTOS para ver los enlaces a contenido reciente.

Introducción a FreeRTOS

> **Important**
>
> Esta página hace referencia al repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial de introducción muestra cómo descargar y configurar FreeRTOS en una máquina host y, a continuación, compilar y ejecutar una aplicación de demostración sencilla en una placa de del microcontrolador calificada.

A lo largo de este tutorial, suponemos que está familiarizado con AWS IoT y la consola de AWS IoT. De lo contrario, le recomendamos que complete el tutorial Introducción a AWS IoT antes de continuar.

Temas:

- Aplicación de demostración FreeRTOS
- Primeros pasos

- Introducción a solución de problemas
- Uso de CMake con FreeRTOS
- Aprovisionamiento de claves en modo desarrollador
- Guías de introducción específicas de placas
- Próximos pasos con FreeRTOS

Aplicación de demostración FreeRTOS

> **Important**
>
> Esta página hace referencia al repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

La aplicación de demostración de este tutorial es la demostración del agente coreMQTT definida en el archivo *freertos*/demos/coreMQTT_Agent/mqtt_agent_task.c. Utiliza la Biblioteca coreMQTT para conectarse a la nube de AWS y publicar periódicamente mensajes en un tema de MQTT alojado en el agente de MQTT de AWS IoT.

Solo se puede ejecutar una aplicación de demostración de FreeRTOS a la vez. Cuando se crea un proyecto de demostración de FreeRTOS, la aplicación que se ejecuta es la primera demostración habilitada en el archivo de encabezado *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h. Para este tutorial, no necesita habilitar o deshabilitar cualquier demostración. La demostración del agente coreMQTT está habilitada de forma predeterminada.

Para obtener más información sobre las aplicaciones de demostración incluidas en FreeRTOS, consulte Demostraciones de FreRTOS.

Primeros pasos

> **Important**
>
> Esta página hace referencia al repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto

FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Para empezar a utilizar FreeRTOS con AWS IoT, necesita una cuenta de AWS, un usuario con permiso para acceder a AWS IoT y servicios en la nube de FreeRTOS. También es necesario que descargue FreeRTOS y configure el proyecto de demostración de FreeRTOS de su placa para que funcione con AWS IoT. Las secciones siguientes le guían a través de estos requisitos.

Note

- Si utilizas el Espressif ESP32- DevKit C, el ESP-WROVER-KIT o el ESP32-WROOM-32SE, omite estos pasos y ve a. Cómo empezar con el Espressif ESP32- C y el ESP-WROVER-KIT DevKit
- Si utiliza Nordic nRF52840-DK, omita estos pasos y vaya a Introducción a Nordic nRF52840-DK.

1. Configuración de su cuenta de AWS y los permisos
2. Registro de la placa de MCU con AWS IoT
3. Descarga de FreeRTOS
4. Configuración de las demostraciones de FreeRTOS

Configuración de su cuenta de AWS y los permisos

Registro para obtener una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Cómo registrarse en 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.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación luego de completar 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.

Crear un usuario administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

Protección de su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Console como propietario de cuenta, elija Usuario raíz e ingrese el email de su Cuenta de AWS. 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 raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario administrativo

1. Activar IAM Identity Center

   Para conocer las instrucciones, consulte Habilitar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center.
2. En IAM Identity Center, otorga acceso administrativo a un usuario administrativo.

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

Cómo iniciar sesión como usuario administrativo

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

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

Para dar acceso, añada permisos a los usuarios, grupos o roles:

- Usuarios y grupos en AWS IAM Identity Center:

  Cree un conjunto de permisos. Siga las instrucciones de Creación de un conjunto de permisos en la Guía del usuario de AWS IAM Identity Center.
- Usuarios administrados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones de Creación de un rol para un proveedor de identidades de terceros (federación) en la Guía del usuario de IAM.
- Usuarios de IAM:
  - Cree un rol que el usuario pueda aceptar. Siga las instrucciones descritas en Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.
  - (No recomendado) Asocie una política directamente a un usuario o añada un usuario a un grupo de usuarios. Siga las instrucciones descritas en Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Registro de la placa de MCU con AWS IoT

Su placa debe estar registrada con AWS IoT para comunicarse con la nube de AWS. Para registrar su placa con AWS IoT, debe tener:

Una política de AWS IoT

La política de AWS IoT concede a su dispositivo permisos para obtener acceso a recursos de AWS IoT. Se almacena en la nube de AWS.

Un objeto de AWS IoT

Un objeto de AWS IoT le permite administrar sus dispositivos en AWS IoT. Se almacena en la nube de AWS.

Una clave privada y un certificado X.509

El certificado y la clave privada permiten a su dispositivo autenticarse en AWS IoT.

Para registrar la placa, siga los procedimientos que se indican a continuación.

Para crear una política de AWS IoT

1. Para crear una política de IAM, debe conocer su región de AWSy número de cuenta de AWS.

   Para encontrar su número de cuenta de AWS, abra la consola de administración de AWS, localice y amplíe el menú situado debajo del nombre de cuenta en la esquina superior derecha y haga clic en Mi cuenta. El ID de su cuenta se muestra en Account Settings (Configuración de cuenta).

   Para buscar la región de AWS para su cuenta de AWS, utilice la AWS Command Line Interface. Para instalar la AWS CLI, siga las instrucciones de la Guía del usuario de AWS Command Line Interface. Después de instalar el AWS CLI, abra una ventana del símbolo del sistema y escriba el siguiente comando:

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

   La salida debe tener el siguiente aspecto:

   ```
   {
       "endpointAddress": "xxxxxxxxxxxxxx-ats.iot.us-west-2.amazonaws.com"
   }
   ```

   En este ejemplo, la región es `us-west-2`.

   > **Note**
   >
   > Recomendamos utilizar los puntos de conexión de ATS, tal y como se muestra en el ejemplo.

2. Vaya a la consola de AWS IoT.
3. En el panel de navegación, elija Secure (Seguridad), elija Policies (Políticas) y, a continuación, elija Create (Crear).

4. Especifique un nombre que identifique la política.

5. En la sección Añadir declaraciones, elija Modo avanzado. Copie y pegue la siguiente política JSON en la ventana del editor de políticas. Sustituya *aws-region* y *aws-account* por su región de AWS y su ID de cuenta.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": "iot:Connect",
        "Resource":"arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
        "Effect": "Allow",
        "Action": "iot:Publish",
        "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
         "Effect": "Allow",
         "Action": "iot:Subscribe",
         "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
         "Effect": "Allow",
         "Action": "iot:Receive",
         "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    }
    ]
}
```

Esta política le concede los siguientes permisos:

`iot:Connect`

Concede a su dispositivo permiso para conectarse con el agente de mensajes de AWS IoT mediante cualquier ID de cliente.

`iot:Publish`

Concede a su dispositivo permiso para publicar un mensaje de MQTT en cualquier tema de MQTT.

`iot:Subscribe`

Concede a su dispositivo permiso para suscribirse a cualquier filtro de temas de MQTT.

`iot:Receive`

Concede a su dispositivo permiso para recibir mensajes del agente de mensajes de AWS IoT acerca de cualquier tema de MQTT.

6. Seleccione Crear.

Para crear un objeto de IoT, clave privada y certificado para su dispositivo.

1. Vaya a la consola de AWS IoT.
2. En el panel de navegación, elija Manage (Administrar) y, a continuación, Things (Cosas).
3. Si no tiene ningún objeto de IoT registrado en su cuenta, se muestra la página Aún no tiene ningún objeto. Si ve esta página, elija Registrar un objeto. De lo contrario, seleccione Crear.
4. En la página Creación de objetos de AWS IoT, elija Crear un solo objeto.
5. En la página Añadir su dispositivo al registro de objetos, escriba un nombre para su objeto y, a continuación, haga clic en Siguiente.
6. En la página Añadir un certificado para el objeto, en Creación de un certificado con un clic, elija Crear certificado.
7. Descargue su clave privada y certificado eligiendo los enlaces Descargar para cada uno de ellos.
8. Elija Activar para activar su certificado. Los certificados deben activarse para poder usarlos.
9. Elija Asociar una política para asociar una política a su certificado que concede al dispositivo acceso a las operaciones de AWS IoT.
10. Elija la política que acaba de crear y elija Registrar objeto.

Una vez que la placa se ha registrado con AWS IoT, puede continuar en Descarga de FreeRTOS.

Descarga de FreeRTOS

Puede descargar FreeRTOS desde el repositorio de FreeRTOS. GitHub

Después de descargar FreeRTOS, puede continuar con Configuración de las demostraciones de FreeRTOS.

Configuración de las demostraciones de FreeRTOS

Tiene que editar algunos archivos de configuración en el directorio de FreeRTOS antes de poder compilar y ejecutar demostraciones en la placa.

Para configurar su punto de enlace de AWS IoT

Debe proporcionar FreeRTOS con su punto de conexión de AWS IoT para que la aplicación que se ejecuta en la placa pueda enviar solicitudes al punto de conexión correcto.

1. Vaya a la consola de AWS IoT.
2. En el panel de navegación izquierdo, elija Configuración.

   Su punto de conexión AWS IoT aparece en Punto de enlace de datos de dispositivo. Debe tener un aspecto similar al siguiente: *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Tome nota de este punto de enlace.
3. En el panel de navegación, elija Manage (Administrar) y, a continuación, Things (Cosas).

   Su dispositivo debe tener un nombre de objeto de AWS IoT. Anote este nombre.
4. Abra `demos/include/aws_clientcredential.h`.
5. Especifique valores para las siguientes constantes:

   - `#define clientcredentialMQTT_BROKER_ENDPOINT "`*Your AWS IoT endpoint*`";`
   - `#define clientcredentialIOT_THING_NAME "`*The AWS IoT thing name of your board*`"`

Para configurar la wifi

Si su placa se conecta a Internet mediante una conexión wifi, tiene que proporcionar FreeRTOS con credenciales wifi para conectarse a la red. Si la placa no es compatible con wifi, puede omitir los pasos que se indican a continuación.

1. `demos/include/aws_clientcredential.h`.
2. Especifique valores para las siguientes constantes de `#define`:

   - `#define clientcredentialWIFI_SSID "`*The SSID for your Wi-Fi network*`"`
   - `#define clientcredentialWIFI_PASSWORD "`*The password for your Wi-Fi network*`"`

- `#define clientcredentialWIFI_SECURITY` *`El tipo de seguridad para su red wifi`*

  Los tipos de seguridad válidos son:

  - `eWiFiSecurityOpen` (abierta, sin seguridad)
  - `eWiFiSecurityWEP` (seguridad WEP)
  - `eWiFiSecurityWPA` (seguridad WPA)
  - `eWiFiSecurityWPA2` (seguridad WPA2)

Para dar formato a sus credenciales de AWS IoT

FreeRTOS necesita el certificado y las claves de AWS IoT asociadas a su objeto registrado y sus políticas de permisos para comunicarse correctamente con AWS IoT en nombre de su dispositivo.

> **Note**
>
> Para configurar sus credenciales de AWS IoT, necesita la clave privada y el certificado que descargó desde la consola de AWS IoT cuando registró su dispositivo. Una vez registrado el dispositivo como un objeto de AWS IoT, puede recuperar los certificados del dispositivo desde la consola de AWS IoT, pero no puede recuperar las claves privadas.

FreeRTOS es un proyecto en lenguaje C, y el certificado y la clave privada deben tener un formato específico para que se puedan añadir al proyecto.

1. En una ventana del navegador, abra `tools/certificate_configuration/CertificateConfigurator.html`.
2. En Archivo PEM del certificado, elija el archivo *`ID`*`-certificate.pem.crt` que ha descargado desde la consola de AWS IoT.
3. En Archivo PEM de clave privada, elija el archivo *`ID`*`-private.pem.key` que ha descargado desde la consola de AWS IoT.
4. Elija Generate and save aws_clientcredential_keys.h (Generar y guardar aws_clientcredential_keys.h) y después guarde el archivo en `demos/include`. Esto sobrescribe el archivo existente en el directorio.

> **Note**
>
> El certificado y la clave privada se incluyen en el código solo para fines de demostración. Las aplicaciones de producción deben almacenar estos archivos en un lugar seguro.

Después de configurar FreeRTOS, puede continuar con la guía de introducción de la placa para configurar el hardware de la plataforma y su entorno de desarrollo de software y, a continuación, compilar y ejecutar la demostración en la placa. Para obtener instrucciones específicas de la placa, consulte la Guías de introducción específicas de placas. La aplicación de demostración que se utiliza en el tutorial de introducción es la demostración de autenticación mutua de coreMQTT, que se encuentra en `demos/coreMQTT/mqtt_demo_mutual_auth.c`.

Introducción a solución de problemas

> **Important**
>
> Esta página hace referencia al repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Los siguientes temas puede ayudarle a solucionar los problemas que encuentre al comenzar con FreeRTOS:

Temas

- Consejos generales de introducción a solución de problemas
- Instalación de un emulador de terminal

Para obtener consejos de solución de problemas específicos de la placa, consulte la guía Introducción a FreeRTOS de su placa.

Consejos generales de introducción a solución de problemas

No aparece ningún mensaje en la consola de AWS IoT después de ejecutar el proyecto de demostración Hello World. ¿Qué tengo que hacer?

Pruebe lo siguiente:

1. Abra una ventana de terminal para ver la salida de registro de la muestra. Esto puede ayudarle a determinar qué va mal.
2. Compruebe que sus credenciales de red son válidas.

Los registros que se muestran en mi terminal al ejecutar una demostración están truncados. ¿Cómo puedo aumentar su longitud?

Aumente el valor de `configLOGGING_MAX_MESSAGE_LENGTH` a 255 en el archivo `FreeRTOSconfig.h` de la demostración que está ejecutando:

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

Instalación de un emulador de terminal

Un emulador de terminal puede ayudarle a diagnosticar problemas o verificar que el código de su dispositivo se ejecuta correctamente. Hay una gran variedad de emuladores de terminal disponibles para Windows, macOS y Linux.

Debe conectar la placa a su equipo antes de intentar establecer una conexión en serie con la placa con un emulador de terminal.

Utilice las siguientes opciones para configurar su emulador de terminal:

Configuración de terminal	Valor
Velocidad en baudios	115200
Datos	8 bits
Paridad	ninguno
Detener	1 bit

Configuración de terminal	Valor
Control de flujo	ninguno

Búsqueda del puerto serie de la placa

Si no conoce el puerto serie de su placa, puede ejecutar uno de los siguientes comandos desde la línea de comandos o terminal para obtener los puertos serie de todos los dispositivos conectados a su equipo host:

Windows

```
chgport
```

Linux

```
ls /dev/tty*
```

macOS

```
ls /dev/cu.*
```

Uso de CMake con FreeRTOS

> **Important**
>
> Esta página hace referencia al repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Puede utilizar CMake para generar archivos de creación de proyectos a partir del código fuente de la aplicación de FreeRTOS y para crear y ejecutar el código fuente.

Puede utilizar un IDE para editar, depurar compilar, instalar y ejecutar código en dispositivos calificados de FreeRTOS. Cada guía de introducción específica de la placa incluye instrucciones para configurar el IDE para una determinada plataforma. Si prefiere trabajar sin un IDE, puede utilizar

otras herramientas de edición y depuración de código de terceros para desarrollar y depurar su código y, a continuación, utilizar CMake para compilar y ejecutar las aplicaciones.

Las siguientes placas son compatibles con CMake:

- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Kit de conectividad de IoT de Infineon XMC4800
- Kit de inicio de AWS IoT de Marvell MW320
- Kit de inicio de AWS IoT de Marvell MW322
- Microchip Curiosity PIC32MZEF Bundle
- Kit de desarrollo DK de Nordic nRF52840
- STMicroelectronicsSTM32L4 Discovery Kit IoT Node
- CC3220SF-LAUNCHXL de Texas Instruments
- Simulador de Microsoft Windows

Consulte los temas a continuación para obtener más información sobre cómo utilizar CMake con FreeRTOS.

Temas

Requisitos previos

Asegúrese de que el equipo host cumple los siguientes requisitos previos antes de continuar:

- La cadena de herramientas de compilación del dispositivo debe ser compatible con el sistema operativo del equipo. CMake es compatible con todas las versiones de Windows, macOS y Linux

  No es posible usar el subsistema Windows para Linux (WSL). Utilice CMake nativo en los equipos Windows.
- Debe tener instalada la versión 3.13 o posterior de CMake.

Puede descargar la distribución binaria de CMake de CMake.org.

> **Note**
>
> Si descarga la distribución binaria de CMake, asegúrese de añadir el ejecutable de CMake a la variable de entorno PATH antes de utilizar CMake desde la línea de comandos.

También puede descargar e instalar CMake mediante un administrador de paquetes, como homebrew, en macOS, o scoop o chocolatey en Windows.

> **Note**
>
> Las versiones de CMake proporcionadas en los administradores de paquetes de numerosas distribuciones de Linux son obsoletas. Si el administrador de paquetes de su distribución no incluye la versión más reciente de CMake, puede probar administradores de paquetes diferentes, como `linuxbrew` o `nix`.

- Debe tener un sistema de compilación nativo compatible.

  CMake puede emplearse con numerosos sistemas de compilación nativos, como GNU Make o Ninja. Tanto Make como Ninja se pueden instalar con administradores de paquetes en Linux, macOS y Windows. Si utiliza Make en Windows, puede instalar una versión independiente de Equation o bien puede instalar MinGW, que incluye a Make.

  > **Note**
  >
  > El ejecutable de Make en MinGW se llama `mingw32-make.exe`, en lugar de `make.exe`.

  Le recomendamos que utilice Ninja, ya que es más rápido que Make y además proporciona soporte nativo a todos los sistemas operativos de escritorio.

Desarrollo de aplicaciones de FreeRTOS con herramientas de edición y depuración de código de terceros

Puede utilizar un editor de código y una extensión de depuración o una herramienta de depuración de terceros para desarrollar aplicaciones para FreeRTOS.

Si, por ejemplo, utiliza Visual Studio Code como su editor de código, puede instalar la extensión de VS Code Cortex-Debug como depurador. Cuando termine de desarrollar la aplicación, puede invocar la herramienta de línea de comandos de CMake para compilar su proyecto a partir de VS Code. Para obtener más información sobre cómo utilizar CMake para crear aplicaciones de FreeRTOS, consulte Creación de FreeRTOS con CMake.

Para la depuración, puede proporcionar un VS Code con una configuración de depuración similar a la siguiente:

```
"configurations": [
    {
        "name": "Cortex Debug",
        "cwd": "${workspaceRoot}",
        "executable": "./build/st/stm32l475_discovery/aws_demos.elf",
        "request": "launch",
        "type": "cortex-debug",
        "servertype": "stutil"
    }
]
```

Creación de FreeRTOS con CMake

De forma predeterminada, la salida de CMake se refiere al sistema operativo donde se ejecuta. Para utilizarlo con compilación cruzada, CMake requiere un archivo de cadena de herramientas, que especifica el compilador que desea utilizar. En FreeRTOS, hay archivos de cadena de herramientas predeterminados disponibles en *freertos*/tools/cmake/toolchains. El método para enviar este archivo a CMake varía en función de si utiliza la interfaz de línea de comandos de CMake o la GUI. Para obtener más información, siga las instrucciones de Generación de archivos de compilación (herramienta de línea de comandos de CMake) que aparecen a continuación. Para obtener más información acerca de la compilación cruzada en CMake, consulte Compilación cruzada en la wiki oficial de CMake.

Para compilar un proyecto basado en CMake

1. Ejecute CMake para generar los archivos de compilación para un sistema de compilación nativo, como Make o Ninja.

   Puede usar la herramienta de línea de comandos de CMake o la GUI de CMake para generar los archivos de compilación para el sistema de compilación nativo.

   Para obtener información sobre cómo generar archivos de creación de FreeRTOS, consulte Generación de archivos de compilación (herramienta de línea de comandos de CMake) y Generación de archivos de compilación (interfaz gráfica de usuario de CMake).
2. Invoque el sistema de compilación nativo para convertir el proyecto en un archivo ejecutable.

   Para obtener información sobre cómo generar archivos de creación de FreeRTOS, consulte Creación de FreeRTOS a partir de los archivos de creación generados.

Generación de archivos de compilación (herramienta de línea de comandos de CMake)

Puede utilizar la herramienta de línea de comandos de CMake (cmake) para generar archivos de creación para FreeRTOS. Para generar los archivos de compilación, tiene que especificar una placa de destino, un compilación y la ubicación de del código fuente y el directorio de compilación.

Puede usar las siguientes opciones para cmake:

- `-DVENDOR` - Especifica la placa de destino.
- `-DCOMPILER` - Especifica el compilador.
- `-S` - Especifica la ubicación del código fuente.
- `-B` - Especifica la ubicación de los archivos de creación generados.

> **Note**
>
> El compilador debe encontrarse en la variable `PATH` del sistema, o bien debe especificar su ubicación.

Por ejemplo, si el proveedor es Texas Instruments, la tarjeta es la CC3220 Launchpad y el compilador es GCC para ARM, puede ejecutar el siguiente comando para compilar los archivos de origen del directorio actual a un directorio denominado *build-directory*:

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory
```

> **Note**
>
> Si utiliza Windows, debe especificar el sistema de compilación nativo, porque CMake utiliza Visual Studio de forma predeterminada. Por ejemplo:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G Ninja
> ```
>
> O bien:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G "MinGW Makefiles"
> ```

Las expresiones regulares `${VENDOR}.*` y `${BOARD}.*` se utilizan para buscar una tarjeta que coincida, por lo que no es necesario indicar el nombre completo del proveedor y la tarjeta en las opciones `BOARD` y `VENDOR`. Basta con un nombre parcial, siempre que solo haya una única coincidencia. Por ejemplo, los comandos siguientes generan los mismos archivos de compilación a partir de la misma fuente:

```
cmake -DVENDOR=ti -DCOMPILER=arm-ti -S . -B build-directory
```

```
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -S . -B build-directory
```

```
cmake -DVENDOR=t -DBOARD=cc -DCOMPILER=arm-ti -S . -B build-directory
```

Puede utilizar la opción `CMAKE_TOOLCHAIN_FILE` si desea utilizar un archivo de cadena de herramientas que no se encuentra en el directorio predeterminado `cmake/toolchains`. Por ejemplo:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -S . -
B build-directory
```

Si el archivo de cadena de herramientas no usa rutas absolutas para el compilador y no ha añadido el compilador a la variable de entorno PATH, puede que CMake no lo encuentre. Para asegurarse de que CMake encuentre el archivo de cadena de herramientas, use la opción AFR_TOOLCHAIN_PATH. Esta opción busca en la ruta especificada para el directorio de la cadena de herramientas y en la subcarpeta de la cadena bajo bin. Por ejemplo:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -
DAFR_TOOLCHAIN_PATH='/path/to/toolchain/' -S . -B build-directory
```

Para habilitar la depuración, establezca en CMAKE_BUILD_TYPE el valor debug. Con esta opción habilitada, CMake añade marcadores de depuración a las opciones de compilación y crea FreeRTOS con símbolos de depuración.

```
# Build with debug symbols
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -DCMAKE_BUILD_TYPE=debug -S . -B build-directory
```

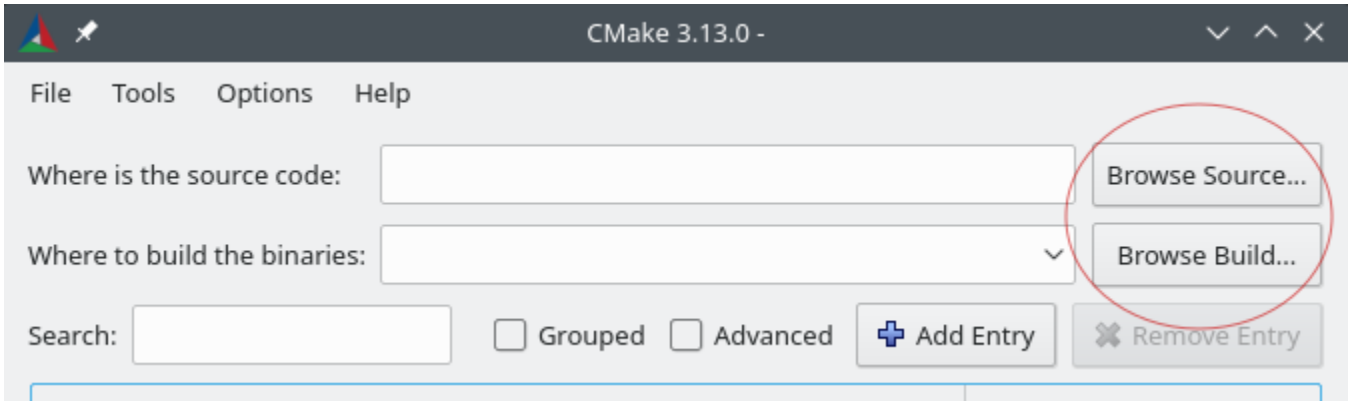
También puede establecer en CMAKE_BUILD_TYPE el valor release para añadir marcadores de optimización a las opciones de compilación.

Generación de archivos de compilación (interfaz gráfica de usuario de CMake)

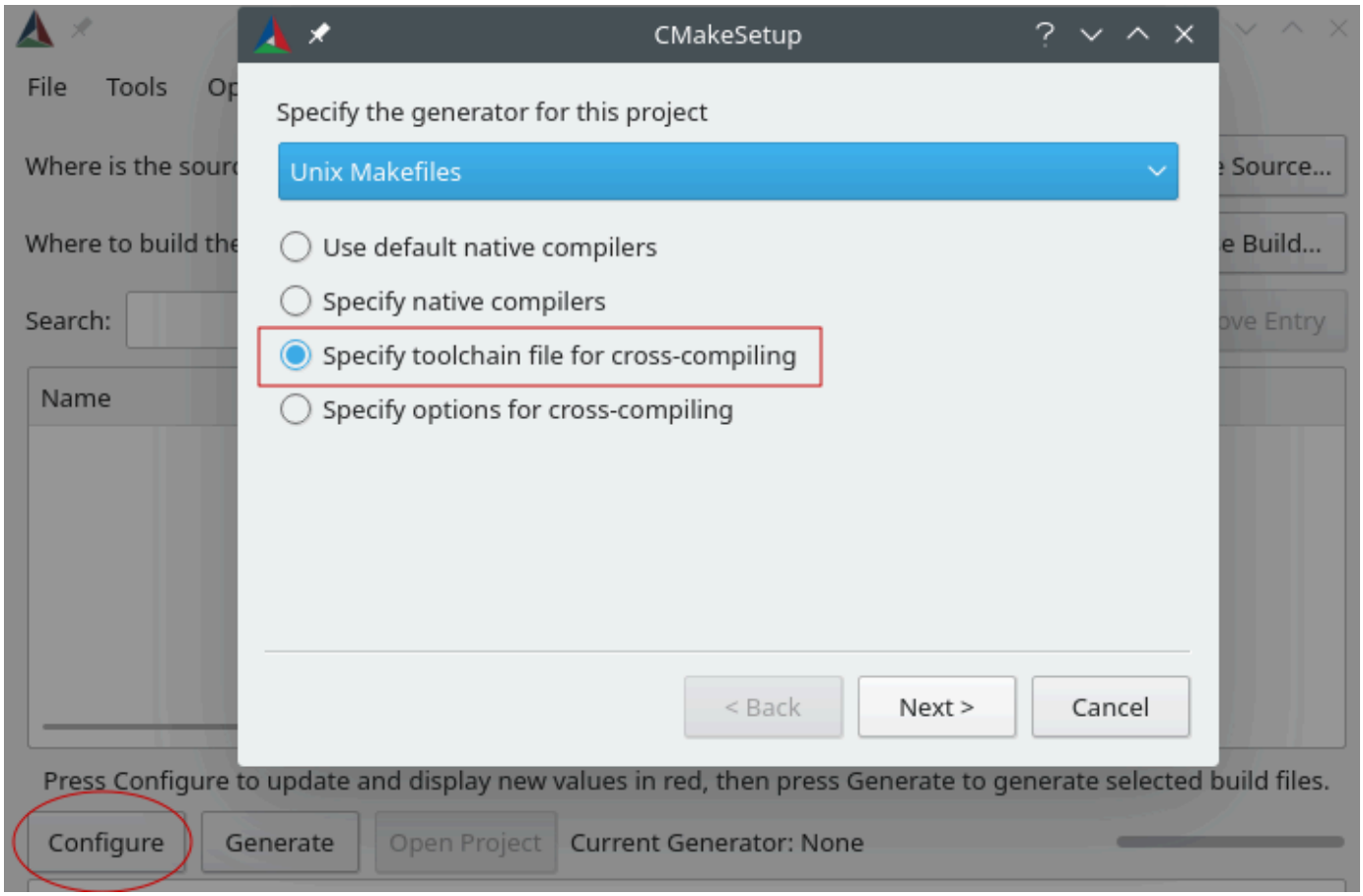
Puede utilizar la GUI de CMake para generar archivos de creación de FreeRTOS.

Para generar los archivos de compilación con la interfaz gráfica de usuario de CMake

1. Desde la línea de comandos, ejecute cmake-gui para iniciar la interfaz gráfica de usuario.
2. Elija Browse Source (Explorar origen), especifique la entrada de origen y, a continuación, elija Browse Build (Explorar compilación) y especifique la salida de la compilación.



3. Elija Configure (Configurar) y busque y elija en Specify the build generator for this project (Especifique el generador de compilación para este proyecto) el sistema de compilación que desee utilizar para compilar los archivos de compilación generados. Si no ve la ventana emergente, es posible que esté reutilizando un directorio de compilación existente. En este caso, elimine la caché de CMake eligiendo Delete Cache (Eliminar caché) en el menú File (Archivo).



4. Elija Specify toolchain file for cross-compiling (Especificar el archivo de cadena de herramientas para la compilación cruzada) y, a continuación, elija Next (Siguiente).

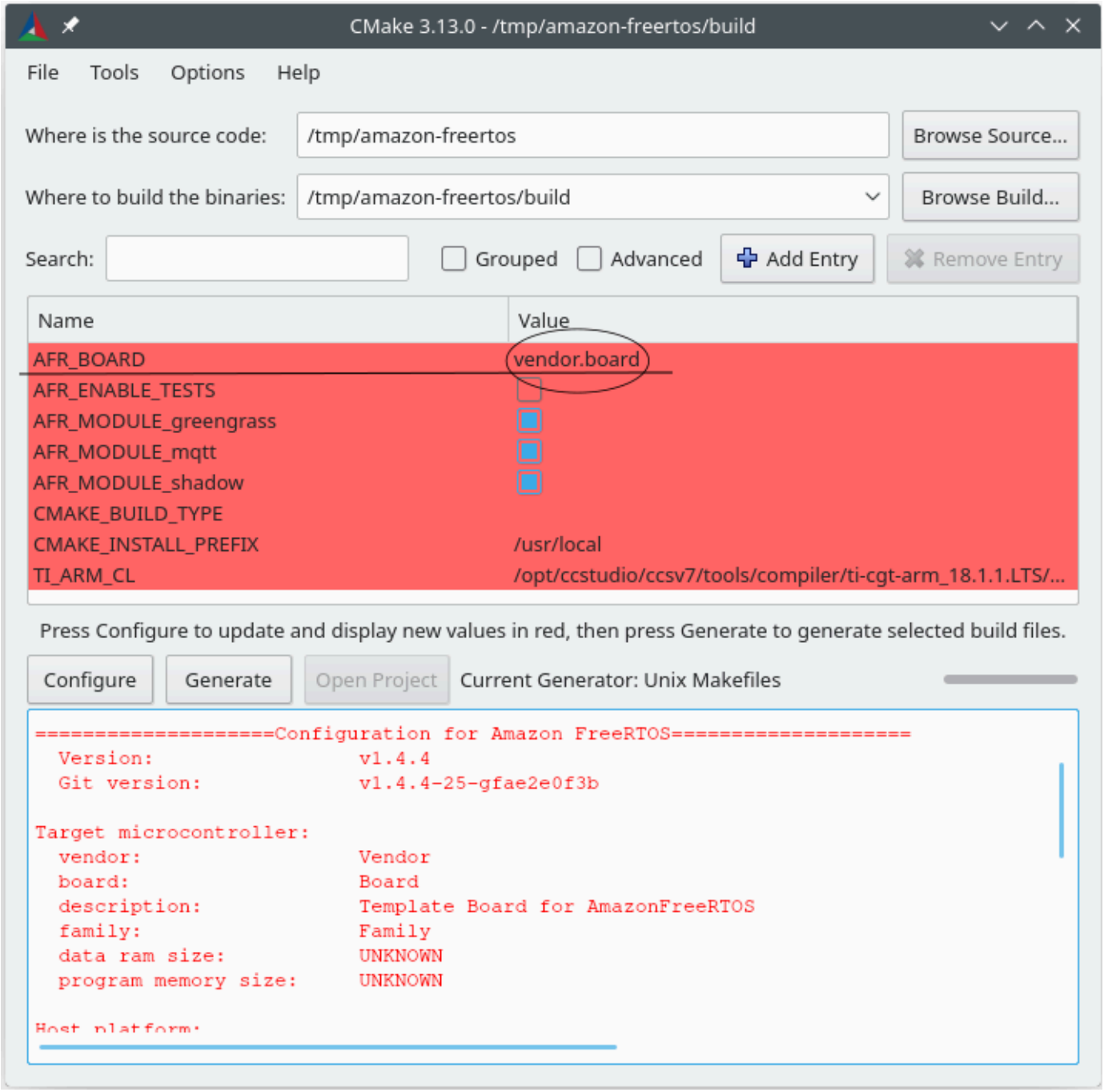
5. Elija el archivo de cadena de herramientas (por ejemplo, *freertos*/tools/cmake/toolchains/arm-ti.cmake) y haga clic en Finalizar.

   La configuración predeterminada para FreeRTOS es la placa de plantilla, que no incluye ningún destino de capa portable. En consecuencia, aparece una ventana con el mensaje Error in configuration process.

   > **Note**
   >
   > Si ve el siguiente error:
   >
   > ```
   > CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):
   > Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
   > ```

   Significa que el compilador no se encuentra en la variable de entorno PATH. Puede configurar la variable AFR_TOOLCHAIN_PATH en la GUI para que indique a CMake dónde ha instalado el compilador. Si no ve la variable AFR_TOOLCHAIN_PATH, elija Add Entry (Añadir entrada). En la ventana emergente, en Name (Nombre), escriba **AFR_TOOLCHAIN_PATH**. En Compiler Path (Ruta del compilador), escriba la ruta hasta su compilador. Por ejemplo, C:/toolchains/arm-none-eabi-gcc.

6. La interfaz gráfica debe tener ahora el aspecto siguiente:



Elija AFR_BOARD, elija la tarjeta y, a continuación, elija de nuevo Configure (Configurar).

7. Elija Generate (Generar). CMake genera los archivos del sistema de compilación (por ejemplo, archivos makefiles o ninja) y estos archivos aparecen en el directorio de compilación que haya especificado en el primer paso. Siga las instrucciones de la siguiente sección para generar la imagen binaria.

Creación de FreeRTOS a partir de los archivos de creación generados

Compilación con un sistema de compilación nativo

Puede crea FreeRTOS con un sistema de creación nativo llamando al comando del sistema de creación desde el directorio de los archivos binarios de salida.

Por ejemplo, si el directorio de salida de los archivos de compilación es `<build_dir>` y utiliza Make como sistema de compilación nativo, ejecute los siguientes comandos:

```
cd <build_dir>
make -j4
```

Compilación de con CMake

También puede utilizar la herramienta de línea de comandos de CMake para crear FreeRTOS. CMake proporciona una capa de abstracción para llamar a sistemas de compilación nativos. Por ejemplo:

```
cmake --build build_dir
```

A continuación se muestran otros usos comunes del modo de compilación de la herramienta de línea de comandos de CMake:

```
# Take advantage of CPU cores.
cmake --build build_dir --parallel 8
```

```
# Build specific targets.
cmake --build build_dir --target afr_kernel
```

```
# Clean first, then build.
cmake --build build_dir --clean-first
```

Para obtener más información acerca del modo de compilación de CMake, consulte la documentación de CMake.

Aprovisionamiento de claves en modo desarrollador

> **Important**
>
> Esta página hace referencia al repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

En esta sección se describen dos opciones para obtener un certificado de cliente X.509 de confianza en un dispositivo IoT para pruebas de laboratorio. Según las funciones del dispositivo, es posible que admita o no varias operaciones relacionadas con el aprovisionamiento, incluida la generación de claves ECDSA integrada, la importación de claves privadas y la inscripción de certificados X.509. Además, diferentes casos de uso requieren diferentes niveles de protección de claves, que van desde el almacenamiento flash integrado hasta el uso de hardware criptográfico específico. Esta sección proporciona la lógica para trabajar en las funciones criptográficas de su dispositivo.

Opción n.º 1: importación de claves privadas desde AWS IoT

Para las pruebas de laboratorio, si su dispositivo permite la importación de claves privadas, siga las instrucciones de Configuración de las demostraciones de FreeRTOS.

Opción n.º 2: generación de claves privadas integrada

Si su dispositivo cuenta con un elemento seguro, o si prefiere generar su propio par de claves del dispositivo y certificado, siga las instrucciones siguientes.

Configuración inicial

1. En primer lugar, siga los pasos de Configuración de las demostraciones de FreeRTOS, pero omita el último paso (es decir, no realice el paso Para formatear sus credenciales de AWS IoT). El resultado neto debe ser la actualización del archivo `demos/include/aws_clientcredential.h` con su configuración, pero no la actualización del archivo `demos/include/aws_clientcredential_keys.h`.

Configuración del proyecto de demostración

Abra la demostración de autenticación mutua de coreMQTT tal y como se describe en la guía de su placa en Guías de introducción específicas de placas. En el proyecto, abra el archivo `aws_dev_mode_key_provisioning.c` y cambie la definición de `keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR`, que está establecida en cero de forma predeterminada, a uno:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

A continuación, cree y ejecute el proyecto de demostración y continúe con el siguiente paso.

Extracción de claves públicas

Dado que el dispositivo aún no se ha aprovisionado con una clave privada y un certificado de cliente, la demostración no se podrá autenticar en AWS IoT. Sin embargo, la demostración de autenticación mutua de coreMQTT comienza ejecutando el aprovisionamiento de claves en modo desarrollador, lo que da lugar a la creación de una clave privada si aún no existe una. Debería ver algo similar a lo siguiente cerca del inicio de la salida de la consola de serie.

```
7 910 [IP-task] Device public key, 91 bytes:
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

Copie las seis líneas de bytes clave en un archivo llamado `DevicePublicKeyAsciiHex.txt`. A continuación, utilice la herramienta de línea de comandos "xxd" para analizar los bytes hexadecimales en binario:

```
xxd -r -ps DevicePublicKeyAsciiHex.txt DevicePublicKeyDer.bin
```

Utilice "openssl" para dar formato a la clave pública del dispositivo codificado binario (DER) como PEM:

```
openssl ec -inform der -in DevicePublicKeyDer.bin -pubin -pubout -outform pem -out
 DevicePublicKey.pem
```

No olvide deshabilitar la configuración de generación de claves temporales que ha habilitado anteriormente. De lo contrario, el dispositivo creará otro par de claves y tendrá que repetir los pasos anteriores:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

Configuración de la infraestructura de claves públicas

Siga las instrucciones de Registro de su certificado de CA para crear una jerarquía de certificados para el certificado de laboratorio de dispositivo. Deténgase antes de ejecutar la secuencia descrita en la sección Creación de un certificado de dispositivo con su certificado de CA.

En este caso, el dispositivo no firmará la solicitud de certificado (es decir, la solicitud de servicio de certificado o CSR), porque la lógica de la codificación X.509 necesaria para crear y firmar una CSR se ha excluido de los proyectos de demostración de FreeRTOS para reducir el tamaño de la ROM. En su lugar, para las pruebas de laboratorio, cree una clave privada en su estación de trabajo y utilícela para firmar la CSR.

```
openssl genrsa -out tempCsrSigner.key 2048
openssl req -new -key tempCsrSigner.key -out deviceCert.csr
```

Una vez que la entidad de certificación se haya creado y registrado con AWS IoT, utilice el siguiente comando para emitir un certificado de cliente basado en la CSR del dispositivo que se ha firmado en el paso anterior:

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key
 -CAcreateserial -out deviceCert.pem -days 500 -sha256 -force_pubkey
 DevicePublicKey.pem
```

Aunque la CSR se haya firmado con una clave privada temporal, el certificado emitido solo se puede utilizar con la clave privada del dispositivo real. El mismo mecanismo se puede utilizar en producción si almacena la clave del firmante de la CSR en hardware independiente y configura la entidad de certificación para que solo emita certificados para las solicitudes firmadas por dicha clave. Esta clave también debe permanecer bajo el control de un administrador designado.

Importación de certificados

Con el certificado emitido, el siguiente paso es importarlo a su dispositivo. También tendrá que importar el certificado de la entidad de certificación (CA), ya que es obligatorio

para que la primera autenticación en AWS IoT se realice correctamente con JITP. En el archivo `aws_clientcredential_keys.h` del proyecto, establezca que la macro `keyCLIENT_CERTIFICATE_PEM` sea el contenido de deviceCert.pem y establezca la macro `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` para que sea el contenido de `rootCA.pem`.

Autorización de dispositivos

Importe `deviceCert.pem` al registro de AWS IoT tal y como se describe en Uso del certificado propio. Debe crear un nuevo objeto de AWS IoT, adjuntar el certificado PENDIENTE y una política a su objeto y, a continuación, marcar el certificado como ACTIVO. Todos estos pasos se pueden realizar manualmente en la consola de AWS IoT.

Una vez que el nuevo certificado de cliente esté ACTIVO y se le haya asociado un objeto y una política, vuelva a ejecutar la demostración de autenticación mutua de coreMQTT. Esta vez, la conexión con el agente MQTT de AWS IoT se realizará correctamente.

Guías de introducción específicas de placas

> **Important**
>
> Esta página hace referencia al repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Después de completar los Primeros pasos, consulte la guía de su placa para obtener instrucciones específicas de la placa sobre cómo empezar a utilizar FreeRTOS:

- Introducción al Kit de desarrollo de Cypress CYW943907AEVAL1F
- Introducción al Kit de desarrollo de Cypress CYW954907AEVAL1F
- Introducción al kit Cypress CY8CKIT-064S0S2-4343W
- Introducción al kit de conectividad de IoT XMC4800 de Infineon
- Introducción al Kit de inicio de AWS IoT MW32x
- Introducción al kit de desarrollo MediaTek MT7697hx
- Introducción a Microchip Curiosity PIC32MZ EF

- Introducción a Nuvoton NuMaker-IoT-M487
- Introducción al módulo de IoT LPC54018 de NXP
- Introducción a Renesas Starter Kit+ for RX65N-2MB
- Introducción al STMicroelectronics STM32L4 Discovery Kit IoT Node
- Introducción a CC3220SF-LAUNCHXL de Texas Instruments
- Introducción al simulador de dispositivos de Windows
- Introducción al Xilinx Avnet MicroZed Industrial IoT Kit

> **Note**
>
> No es necesario que realice los Primeros pasos para las siguientes guías de introducción a FreeRTOS autónomas:
>
> - Introducción al microchip ATECC608A Secure Element con simulador de Windows
> - Cómo empezar con el Espressif ESP32- C y el ESP-WROVER-KIT DevKit
> - Introducción a Espressif ESP32-WROOM-32SE
> - Introducción a Espressif ESP32-S2
> - Introducción al kit de conectividad de IoT XMC4800 y OPTIGA Trust X de Infineon
> - Introducción a Nordic nRF52840-DK

Introducción al Kit de desarrollo de Cypress CYW943907AEVAL1F

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción al kit de desarrollo de Cypress CYW943907AEVAL1F. Si no tiene el kit de desarrollo Cypress CYW943907AEVAL1F, visite AWS Partner Device Catalog para comprar uno de nuestro socio.

> **Note**
>
> Este tutorial presenta la configuración y ejecución de la demostración de autenticación mutua de coreMQTT. Actualmente, el puerto de FreeRTOS para esta placa no admite las demostraciones de servidor TCP y cliente.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos.

> **Important**
>
> - En este tema, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.
> - Los caracteres de espacio en la ruta *freertos* pueden causar errores de compilación. Al clonar o copiar el repositorio, asegúrese de que la ruta que crea no contiene caracteres de espacio.
> - La longitud máxima de una ruta de archivo en Microsoft Windows es de 260 caracteres. Las rutas largas al directorio de descargas de FreeRTOS pueden provocar errores de creación.
> - Como el código fuente puede contener enlaces simbólicos, si utiliza Windows para extraer el archivo, es posible que tenga que:
>   - Habilitar el modo de desarrollador o
>   - Utilizar una consola que tenga el rango de administrador.
>
>   De esta forma, Windows puede crear correctamente enlaces simbólicos al extraer el archivo. De lo contrario, los enlaces simbólicos se escribirán como archivos normales que contengan las rutas de los enlaces simbólicos como texto o estarán vacíos. Para obtener más información, consulte la entrada del blog Symlinks in Windows 10.
>
>   Si usa Git en Windows, debe habilitar el modo desarrollador o debe:
>   - Establecer `core.symlinks` en verdadero con el siguiente comando:
>
>     ```
>     git config --global core.symlinks true
>     ```
>   - Usar una consola que tenga el rango de administrador siempre que utilice un comando git que escriba en el sistema (por ejemplogit pull, git clone y git submodule update --init --recursive).

- Como se indica en Descarga de FreeRTOS, actualmente, los puertos de FreeRTOS para Cypress solo están disponibles en GitHub.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
2. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
3. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.
4. Interacción con la aplicación que se ejecuta en la placa con una conexión serie para fines de monitorización y depuración.

Configuración del entorno de desarrollo de

Descarga e instalación del SDK de WICED Studio.

En esta guía de introducción, se utiliza el SDK de WICED Studio de Cypress para programar la placa con la demostración de FreeRTOS. Visite el sitio web WICED Software para descargar el SDK de WICED Studio de Cypress. Debe registrarse para obtener una cuenta gratuita de Cypress para descargar el software. El SDK de WICED Studio es compatible con los sistemas operativos Windows, macOS y Linux.

> **Note**
>
> Algunos sistemas operativos necesitan pasos de instalación adicionales. Asegúrese de leer y seguir todas las instrucciones de instalación para el sistema operativo y la versión de WICED Studio que está instalando.

Configuración de las variables de entorno

Antes de utilizar WICED Studio para programar la placa, debe crear una variable de entorno para el directorio de instalación del SDK de WICED Studio. Si WICED Studio se está ejecutando mientras crea las variables, debe reiniciar la aplicación después de crearlas.

> **Note**
>
> El instalador de WICED Studio crea dos carpetas separadas denominadas `WICED-Studio-`*`m`*`.`*`n`* en su equipo, donde `m` y `n` son los números de versión principal y secundaria, respectivamente. En este documento se asume un nombre de carpeta de `WICED-Studio-6.2` pero asegúrese de utilizar el nombre correcto para la versión que instale. Cuando defina la variable de entorno ,`WICED_STUDIO_SDK_PATH` asegúrese de especificar la ruta de instalación completa del SDK de WICED Studio y no la ruta de instalación de la interfaz de usuario de WICED Studio. En Windows y macOS, la carpeta `WICED-Studio-`*`m`*`.`*`n`* para el SDK se crea en la carpeta `Documents` de forma predeterminada.

Para crear la variable de entorno en Windows

1. Abra el Panel de control, elija Sistema y, a continuación, elija Configuración avanzada del sistema.
2. En la pestaña Opciones avanzadas, elija Variables de entorno.
3. En Variables de usuario, elija Nueva.
4. En Nombre de la variable, escriba **`WICED_STUDIO_SDK_PATH`**. En Valor de la variable, especifique el directorio de instalación del SDK de WICED Studio.

Para crear la variable de entorno en Linux o macOS

1. Abra el archivo `/etc/profile` en su equipo y añada lo siguiente a la última línea del archivo:

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. Reinicie el equipo.
3. Abra una ventana de terminal y ejecute los siguientes comandos:

   ```
   cd freertos/vendors/cypress/WICED_SDK
   ```

   ```
   perl platform_adjust_make.pl
   ```

   ```
   chmod +x make
   ```

Establecimiento de una conexión serie

Para establecer una conexión serie entre la máquina host y la placa

1. Conecte la placa al equipo host con un cable USB estándar-A a micro-B.
2. Identifique el número de puerto serie USB para la conexión a la placa en el equipo host.
3. Inicie un terminal serie y abra una conexión con los siguientes valores de configuración:
   - Velocidad en baudios: 115 200
   - Datos: 8 bits
   - Paridad: ninguna
   - Bits de parada: 1
   - Control del flujo: ninguno

Para obtener más información acerca de cómo instalar un terminal y configurar una conexión serie, consulte Instalación de un emulador de terminal.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Creación y ejecución del proyecto de demostración de FreeRTOS

Después de configurar una conexión en serie para la placa, puede crear el proyecto de demostración de FreeRTOS, instalar la aplicación de demostración en la placa y, a continuación, ejecutar la demostración.

Para crear y ejecutar el proyecto de demostración de FreeRTOS en WICED Studio

1. Lance WICED Studio.
2. En el menú Archivo, elija Importar. Expanda la carpeta `General`, elija Existing Projects into Workspace (Proyectos existentes a Workspace) y, a continuación, elija Next (Siguiente).
3. En Select root directory (Seleccionar directorio raíz), seleccione Browse... (Examinar...), vaya a la ruta *`freertos`*`/projects/cypress/CYW943907AEVAL1F/wicedstudio` y, a continuación, seleccione OK (Aceptar).
4. En Projects (Proyectos), marque la casilla solo del proyecto aws_demo . Elija Finish (Finalizar) para importar el proyecto. El proyecto de destino aws_demo debe aparecer en la ventana Make Target (Hacer objetivo).
5. Amplíe el menú WICED Platform (Plataforma WICED) y elija WICED Filters off (Desactivar filtros de WICED).
6. En la ventana Make Target (Hacer objetivo), amplía aws_demo, haga clic con el botón derecho en el archivo `demo.aws_demo` y, a continuación, elija Build Target (Compilar objetivo) para compilar y descargar la demostración en la placa. La demostración debe ejecutarse automáticamente después de compilarla y descargarla en la placa.

Solución de problemas

- Si utiliza Windows, es posible que reciba el siguiente error al compilar y ejecutar el proyecto de demostración:

  ```
  : recipe for target 'download_dct' failed
  make.exe[1]: *** [download_dct] Error 1
  ```

  Para solucionar este error, haga lo siguiente:

  1. Desplácese hasta *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32` y haga doble clic en `openocd-all-brcm-libftdi.exe`.
  2. Desplácese hasta *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1` y haga doble clic en `InstallDriver.exe`.
- Si utiliza Linux o macOS, es posible que reciba el siguiente error al compilar y ejecutar el proyecto de demostración:

  ```
  make[1]: *** [download_dct] Error 127
  ```

Para solucionar este error, utilice el siguiente comando para actualizar el paquete libusb-dev:

```
sudo apt-get install libusb-dev
```

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción al Kit de desarrollo de Cypress CYW954907AEVAL1F

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial proporciona instrucciones para empezar a trabajar con el kit de desarrollo de Cypress CYW954907AEVAL1F. Si no tiene todavía el kit de desarrollo Cypress CYW954907AEVAL1F, visite AWS Partner Device Catalog para comprar uno de nuestro socio.

> **Note**
>
> Este tutorial presenta la configuración y ejecución de la demostración de autenticación mutua de coreMQTT. Actualmente, el puerto de FreeRTOS para esta placa no admite las demostraciones del servidor TCP y del cliente.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.

> **Important**
>
> - En este tema, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.

- Los caracteres de espacio en la ruta *freertos* pueden causar errores de compilación. Al clonar o copiar el repositorio, asegúrese de que la ruta que crea no contiene caracteres de espacio.
- La longitud máxima de una ruta de archivo en Microsoft Windows es de 260 caracteres. Las rutas largas al directorio de descargas de FreeRTOS pueden provocar errores de creación.
- Como el código fuente puede contener enlaces simbólicos, si utiliza Windows para extraer el archivo, es posible que tenga que:
  - Habilitar el modo de desarrollador o
  - Utilizar una consola que tenga el rango de administrador.

  De esta forma, Windows puede crear correctamente enlaces simbólicos al extraer el archivo. De lo contrario, los enlaces simbólicos se escribirán como archivos normales que contengan las rutas de los enlaces simbólicos como texto o estarán vacíos. Para obtener más información, consulte la entrada del blog Symlinks in Windows 10.

  Si usa Git en Windows, debe habilitar el modo desarrollador o debe:
  - Establecer `core.symlinks` en verdadero con el siguiente comando:

    ```
    git config --global core.symlinks true
    ```
  - Usar una consola que tenga el rango de administrador siempre que utilice un comando git que escriba en el sistema (por ejemplogit pull, git clone y git submodule update --init --recursive).
- Como se indica en Descarga de FreeRTOS, actualmente, los puertos de FreeRTOS para Cypress solo están disponibles en GitHub.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
2. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
3. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.

4. Interacción con la aplicación que se ejecuta en la placa con una conexión serie para fines de monitorización y depuración.

Configuración del entorno de desarrollo de

Descarga e instalación del SDK de WICED Studio.

En esta guía de introducción, se utiliza el SDK de WICED Studio de Cypress para programar la placa con la demostración de FreeRTOS. Visite el sitio web WICED Software para descargar el SDK de WICED Studio de Cypress. Debe registrarse para obtener una cuenta gratuita de Cypress para descargar el software. El SDK de WICED Studio es compatible con los sistemas operativos Windows, macOS y Linux.

> **Note**
>
> Algunos sistemas operativos necesitan pasos de instalación adicionales. Asegúrese de leer y seguir todas las instrucciones de instalación para el sistema operativo y la versión de WICED Studio que está instalando.

Configuración de las variables de entorno

Antes de utilizar WICED Studio para programar la placa, debe crear una variable de entorno para el directorio de instalación del SDK de WICED Studio. Si WICED Studio se está ejecutando mientras crea las variables, debe reiniciar la aplicación después de crearlas.

> **Note**
>
> El instalador de WICED Studio crea dos carpetas separadas denominadas `WICED-Studio-`*`m`*`.`*`n`* en su equipo, donde `m` y `n` son los números de versión principal y secundaria, respectivamente. En este documento se asume un nombre de carpeta de `WICED-Studio-6.2` pero asegúrese de utilizar el nombre correcto para la versión que instale. Cuando defina la variable de entorno ,`WICED_STUDIO_SDK_PATH` asegúrese de especificar la ruta de instalación completa del SDK de WICED Studio y no la ruta de instalación de la interfaz de usuario de WICED Studio. En Windows y macOS, la carpeta `WICED-Studio-`*`m`*`.`*`n`* para el SDK se crea en la carpeta `Documents` de forma predeterminada.

Para crear la variable de entorno en Windows

1. Abra el Panel de control, elija Sistema y, a continuación, elija Configuración avanzada del sistema.
2. En la pestaña Opciones avanzadas, elija Variables de entorno.
3. En Variables de usuario, elija Nueva.
4. En Nombre de la variable, escriba **`WICED_STUDIO_SDK_PATH`**. En Valor de la variable, especifique el directorio de instalación del SDK de WICED Studio.

Para crear la variable de entorno en Linux o macOS

1. Abra el archivo `/etc/profile` en su equipo y añada lo siguiente a la última línea del archivo:

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. Reinicie el equipo.
3. Abra una ventana de terminal y ejecute los siguientes comandos:

   ```
   cd freertos/vendors/cypress/WICED_SDK
   ```

   ```
   perl platform_adjust_make.pl
   ```

   ```
   chmod +x make
   ```

Establecimiento de una conexión serie

Para establecer una conexión serie entre la máquina host y la placa

1. Conecte la placa al equipo host con un cable USB estándar-A a micro-B.
2. Identifique el número de puerto serie USB para la conexión a la placa en el equipo host.
3. Inicie un terminal serie y abra una conexión con los siguientes valores de configuración:

   - Velocidad en baudios: 115 200
   - Datos: 8 bits
   - Paridad: ninguna

- Bits de parada: 1
- Control del flujo: ninguno

Para obtener más información acerca de cómo instalar un terminal y configurar una conexión serie, consulte Instalación de un emulador de terminal.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Creación y ejecución del proyecto de demostración de FreeRTOS

Después de configurar una conexión en serie para la placa, puede crear el proyecto de demostración de FreeRTOS, instalar la aplicación de demostración en la placa y, a continuación, ejecutar la demostración.

Para crear y ejecutar el proyecto de demostración de FreeRTOS en WICED Studio

1. Lance WICED Studio.
2. En el menú Archivo, elija Importar. Expanda la carpeta `General`, elija Existing Projects into Workspace (Proyectos existentes a Workspace) y, a continuación, elija Next (Siguiente).
3. En Select root directory (Seleccionar directorio raíz), seleccione Browse... (Examinar...), vaya a la ruta *`freertos`*`/projects/cypress/CYW954907AEVAL1F/wicedstudio` y, a continuación, seleccione OK (Aceptar).
4. En Projects (Proyectos), marque la casilla solo del proyecto aws_demo . Elija Finish (Finalizar) para importar el proyecto. El proyecto de destino aws_demo debe aparecer en la ventana Make Target (Hacer objetivo).

5. Amplíe el menú WICED Platform (Plataforma WICED) y elija WICED Filters off (Desactivar filtros de WICED).
6. En la ventana Make Target (Hacer objetivo), amplía aws_demo, haga clic con el botón derecho en el archivo `demo.aws_demo` y, a continuación, elija Build Target (Compilar objetivo) para compilar y descargar la demostración en la placa. La demostración debe ejecutarse automáticamente después de compilarla y descargarla en la placa.

Solución de problemas

- Si utiliza Windows, es posible que reciba el siguiente error al compilar y ejecutar el proyecto de demostración:

  ```
  : recipe for target 'download_dct' failed
  make.exe[1]: *** [download_dct] Error 1
  ```

  Para solucionar este error, haga lo siguiente:

  1. Desplácese hasta *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32` y haga doble clic en `openocd-all-brcm-libftdi.exe`.
  2. Desplácese hasta *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1` y haga doble clic en `InstallDriver.exe`.
- Si utiliza Linux o macOS, es posible que reciba el siguiente error al compilar y ejecutar el proyecto de demostración:

  ```
  make[1]: *** [download_dct] Error 127
  ```

  Para solucionar este error, utilice el siguiente comando para actualizar el paquete libusb-dev:

  ```
  sudo apt-get install libusb-dev
  ```

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción al kit Cypress CY8CKIT-064S0S2-4343W

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial proporciona instrucciones para empezar a utilizar el kit CY8CKIT-064S0S2-4343W. Si aún no lo tiene, puede usar ese enlace para adquirir un kit. También puede usar ese enlace para acceder a la guía del usuario del kit.

Introducción

Antes de comenzar, debe configurar AWS IoT y FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. Después de completar los requisitos previos, tendrá un paquete de FreeRTOS con credenciales de AWS IoT Core.

> **Note**
>
> En este tutorial, la ruta al directorio de descargas de FreeRTOS creado en la sección “Primeros pasos” se denomina *`freertos`*.

Configuración del entorno de desarrollo

FreeRTOS funciona con un flujo de creación de CMake o Make. Puede usar ModusToolbox para su flujo de creación de Make. Puede usar el IDE de Eclipse que se entrega con ModusToolbox o un IDE asociado, como IAR EW-Arm, Arm MDK o Microsoft Visual Studio Code. El IDE de Eclipse es compatible con los sistemas operativos Windows, macOS y Linux.

Antes de empezar, descargue e instale la última versión del software ModusToolbox. Para obtener más información, consulte la Guía de instalación de ModusToolbox.

Herramientas de actualización para ModusToolbox 2.1 o versiones anteriores

Si utiliza el IDE ModusToolbox 2.1 de Eclipse para programar este kit, tendrá que actualizar las herramientas OpenOCD y Firmware-Loader.

En los siguientes pasos, la ruta *ModusToolbox* predeterminada para:

- Windows es `C:\Users\`*user_name*`\ModusToolbox`.
- Linux es *user_home*`/ModusToolbox` o la ubicación donde elija extraer el archivo comprimido.
- MacOS se encuentra en la carpeta Aplicaciones del volumen que seleccione en el asistente.

Actualización de OpenOCD

Este kit requiere Cypress OpenOCD 4.0.0 o posterior para borrar y programar correctamente el chip.

Para actualizar Cypress OpenOCD

1. Vaya a la página de la versión de Cypress OpenOCD.
2. Descargue el archivo de almacenamiento para su sistema operativo (Windows/Mac/Linux).
3. Elimine los archivos existentes en *ModusToolbox*`/tools_2.x/openocd`.
4. Sustituya los archivos de *ModusToolbox*`/tools_2.x/openocd` por el contenido extraído del archivo que descargó en un paso anterior.

Actualización de Firmware-loader

Este kit requiere Cypress Firmware-Loader 3.0.0 o una versión posterior.

Para actualizar Cypress Firmware-Loader

1. Vaya a la página de la versión de Cypress Firmware-loader.
2. Descargue el archivo de almacenamiento para su sistema operativo (Windows/Mac/Linux).
3. Elimine los archivos existentes en *ModusToolbox*`/tools_2.x/fw-loader`.
4. Sustituya los archivos de *ModusToolbox*`/tools_2.x/fw-loader` por el contenido extraído del archivo que descargó en un paso anterior.

Como alternativa, puede usar CMake para generar archivos de creación del proyecto a partir del código fuente de la aplicación FreeRTOS, crear el proyecto con su herramienta de creación preferida y luego programar el kit con OpenOCD. Si prefiere usar una herramienta de interfaz gráfica de usuario para programar con el flujo de CMake, descargue e instale Cypress Programmer desde la página web de Cypress Programming Solutions. Para obtener más información, consulte Uso de CMake con FreeRTOS.

Configurar su hardware

Siga estos pasos para configurar el hardware del kit.

1. Aprovisione el kit

   Siga las instrucciones de la Guía de aprovisionamiento del kit CY8CKIT-064S0S2-4343W para aprovisionar el kit a AWS IoT de forma segura.

   Este kit requiere CySecureTools 3.1.0 o posterior.

2. Configure una conexión serie

   a. Conecte el kit al equipo host.

   b. El puerto serie USB del kit se enumera automáticamente en el equipo host. Identifique el número de puerto. En Windows, puede identificarlo mediante el Administrador de dispositivos en Puertos (COM y LPT).

   c. Inicie un terminal serie y abra una conexión con los siguientes valores de configuración:

      - Velocidad en baudios: 115 200
      - Datos: 8 bits
      - Paridad: ninguna
      - Bits de parada: 1
      - Control del flujo: ninguno

Creación y ejecución del proyecto de demostración de FreeRTOS

En esta sección, creará y ejecutará la demostración.

1. Asegúrese de seguir los pasos de la Guía de aprovisionamiento del kit CY8CKIT-064S0S2-4343W.
2. Cree la demostración de FreeRTOS.

   a. Abra el IDE de Eclipse para ModusToolbox y elija o cree un espacio de trabajo.

   b. En el menú Archivo, elija Importar.

      Expanda General, elija Proyectos existentes en el espacio de trabajo y, a continuación, elija Siguiente.

c. En Directorio raíz, introduzca *freertos*/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos y, a continuación, seleccione el nombre del proyecto aws_demos. Debería seleccionarse de forma predeterminada.

d. Elija Finalizar para importar el proyecto en su espacio de trabajo.

e. Cree la aplicación realizando una de las siguientes operaciones:

   - En Panel rápido, seleccione Crear la aplicación aws_demos.
   - Elija Proyecto y, a continuación, elija Crear todo.

   Asegúrese de que el proyecto se crea sin errores.

3. Monitorización de mensajes de MQTT en la nube

   Antes de ejecutar la demostración, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS. Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT, siga estos pasos.

   a. Inicie sesión en la consola de AWS IoT.

   b. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.

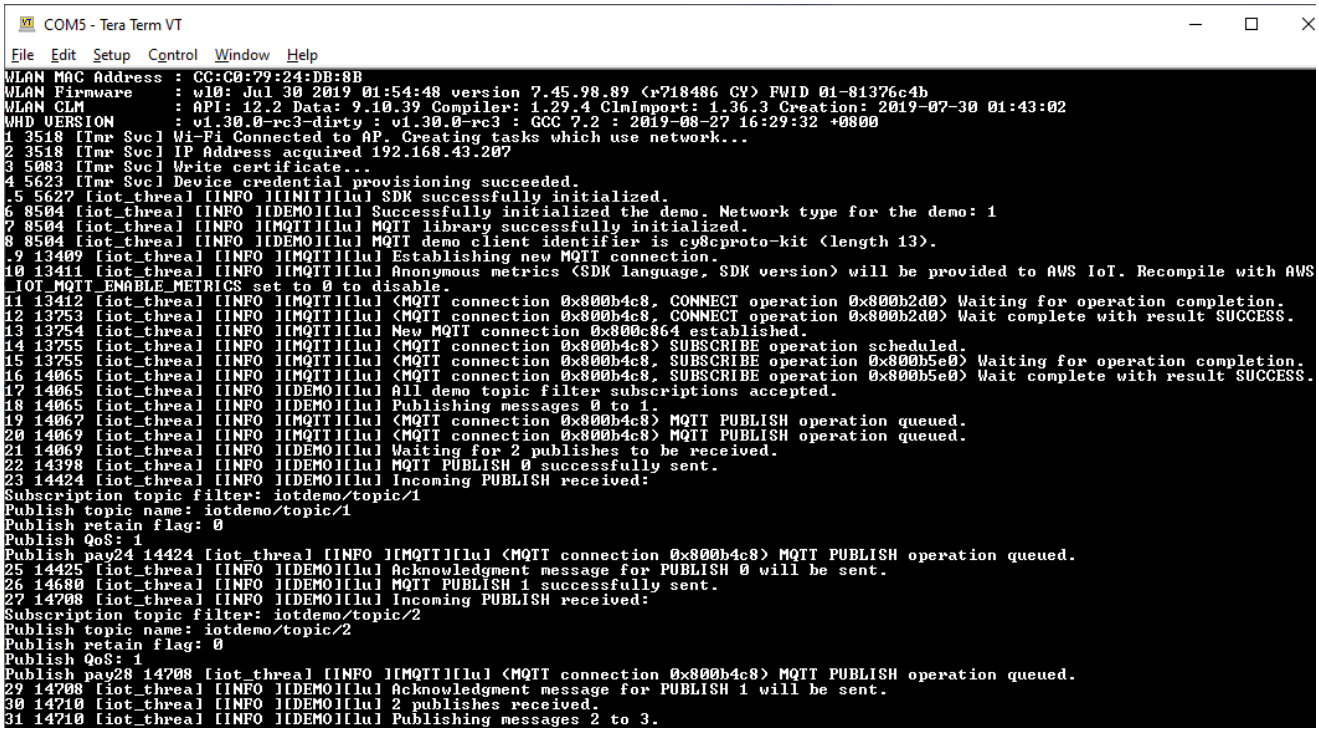
   c. En Tema de suscripción, escriba ***your-thing-name*/example/topic** y, a continuación, elija Suscribirse al tema.

4. Ejecute el proyecto de demostración de FreeRTOS

   a. Seleccione el proyecto aws_demos en el espacio de trabajo.

   b. En Panel rápido, seleccione el programa aws_demos (KitProg3). Esto programa la placa y la aplicación de demostración comienza a ejecutarse una vez finalizada la programación.

   c. Puede ver el estado de la aplicación que se está ejecutando en el terminal serie. La siguiente figura muestra una parte de la salida del terminal.



La demostración de MQTT publica mensajes sobre cuatro temas diferentes (`iotdemo/topic/`*n*, donde n=1 a 4) y se suscribe a todos esos temas para recibir los mismos mensajes. Cuando se recibe un mensaje, la demostración publica un mensaje de acuse de recibo sobre el tema `iotdemo/acknowledgements`. La siguiente lista describe los mensajes de depuración que aparecen en la salida del terminal, con referencias a los números de serie de los mensajes. En la salida, los detalles del controlador WICED Host Driver (WHD) se imprimen primero sin numeración de serie.

1. 1 al 4 - El dispositivo se conecta al punto de acceso (AP) configurado y se aprovisiona conectándose al servidor AWS mediante el punto de conexión y los certificados configurados.
2. 5 a 13 - La biblioteca coreMQTT se inicializa y el dispositivo establece la conexión MQTT.
3. 14 a 17 - El dispositivo se suscribe a todos los temas para recibir los mensajes publicados.
4. 18 a 30 - El dispositivo publica dos mensajes y espera a recibirlos de vuelta. Cuando se recibe cada mensaje, el dispositivo envía un mensaje de acuse de recibo.

El mismo ciclo de publicación, recepción y acuse de recibo continúa hasta que se publiquen todos los mensajes. Se publican dos mensajes por ciclo hasta completar el número de ciclos configurado.

5. Uso de CMake con FreeRTOS

   También puede utilizar CMake para crear y ejecutar la aplicación de demostración. Para configurar CMake y un sistema de compilación nativo, consulte Requisitos previos.

   a. Utilice el siguiente comando para generar archivos de creación. Especifique la placa de destino con la opción `-DBOARD`.

   ```
   cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
   S freertos -B build_dir
   ```

   Si utiliza Windows, debe especificar el sistema de creación nativo con la opción `-G` porque CMake utiliza Visual Studio de forma predeterminada.

   Example

   ```
   cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
   S freertos -B build_dir -G Ninja
   ```

   Si `arm-none-eabi-gcc` no se encuentra en la ruta del shell, también debe configurar la variable `AFR_TOOLCHAIN_PATH` de CMake.

   Example

   ```
    -DAFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
   ```

   b. Utilice el siguiente comando para crear el proyecto con CMake.

   ```
   cmake --build build_dir
   ```

   c. Por último, programe los archivos `cm0.hex` y `cm4.hex` generados en *`build_dir`* mediante Cypress Programmer.

Ejecución de otras demostraciones

Se ha probado y verificado que las siguientes aplicaciones de demostración funcionan con la versión actual. Puede encontrar estas demostraciones en el directorio *`freertos`*`/demos`. Para obtener información sobre cómo ejecutar estas demostraciones, consulte Demostraciones de FreRTOS.

- Demostración de Bluetooth de bajo consumo

- Demostración de actualizaciones inalámbricas
- Demostración de cliente de Echo de sockets seguros
- Demostración de sombra de dispositivo de AWS IoT

Debugging

El KitProg3 del kit admite la depuración mediante el protocolo SWD.

- Para depurar la aplicación FreeRTOS, seleccione el proyecto aws_demos en el espacio de trabajo y, a continuación, seleccione Depuración de aws_demos (KitProg3) en Panel rápido.

Actualizaciones OTA

Los MCU PSoC 64 han superado todas las pruebas de calificación de FreeRTOS requeridas. Sin embargo, la característica inalámbrica (OTA) opcional implementada en la biblioteca de firmware de AWS PSoC 64 Standard Secure aún está pendiente de evaluación. La característica OTA, tal como está implementada, supera actualmente todas las pruebas de calificación de OTA, excepto la de aws_ota_test_case_rollback_if_unable_to_connect_after_update.py.

Cuando una imagen OTA validada correctamente se aplica a un dispositivo mediante el estándar PSoC64 Standard Secure - MCU de AWS y el dispositivo no puede comunicarse con AWS IoT Core, el dispositivo no puede revertir automáticamente a la imagen original conocida correcta. Esto puede provocar que no se pueda acceder al dispositivo desde AWS IoT Core para realizar más actualizaciones. El equipo de Cypress aún está desarrollando esta funcionalidad.

Para obtener más información, consulte Actualizaciones OTA con AWS y el kit CY8CKIT-064S0S2-4343W. Si tiene más preguntas o necesita asistencia técnica, póngase en contacto con la Comunidad de desarrolladores de Cypress.

Introducción al microchip ATECC608A Secure Element con simulador de Windows

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para empezar a usar el microchip ATECC608A Secure Element con simulador de Windows.

Necesitará el siguiente hardware:

- Microchip ATECC608A Secure Element clickboard
- SAMD21 XPlained Pro
- Adaptador mikroBUS Xplained Pro

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.

Información general

Este tutorial contiene los siguientes pasos:

1. Conecte la placa a un equipo host.
2. Instale el software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
3. Realice la compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
4. Cargue la imagen binaria de la aplicación en su placa y, a continuación, ejecute la aplicación.

Configuración del hardware del microchip ATECC608A

Para poder interactuar con el dispositivo Microchip ATECC608A, debe programar primero la placa SAMD21.

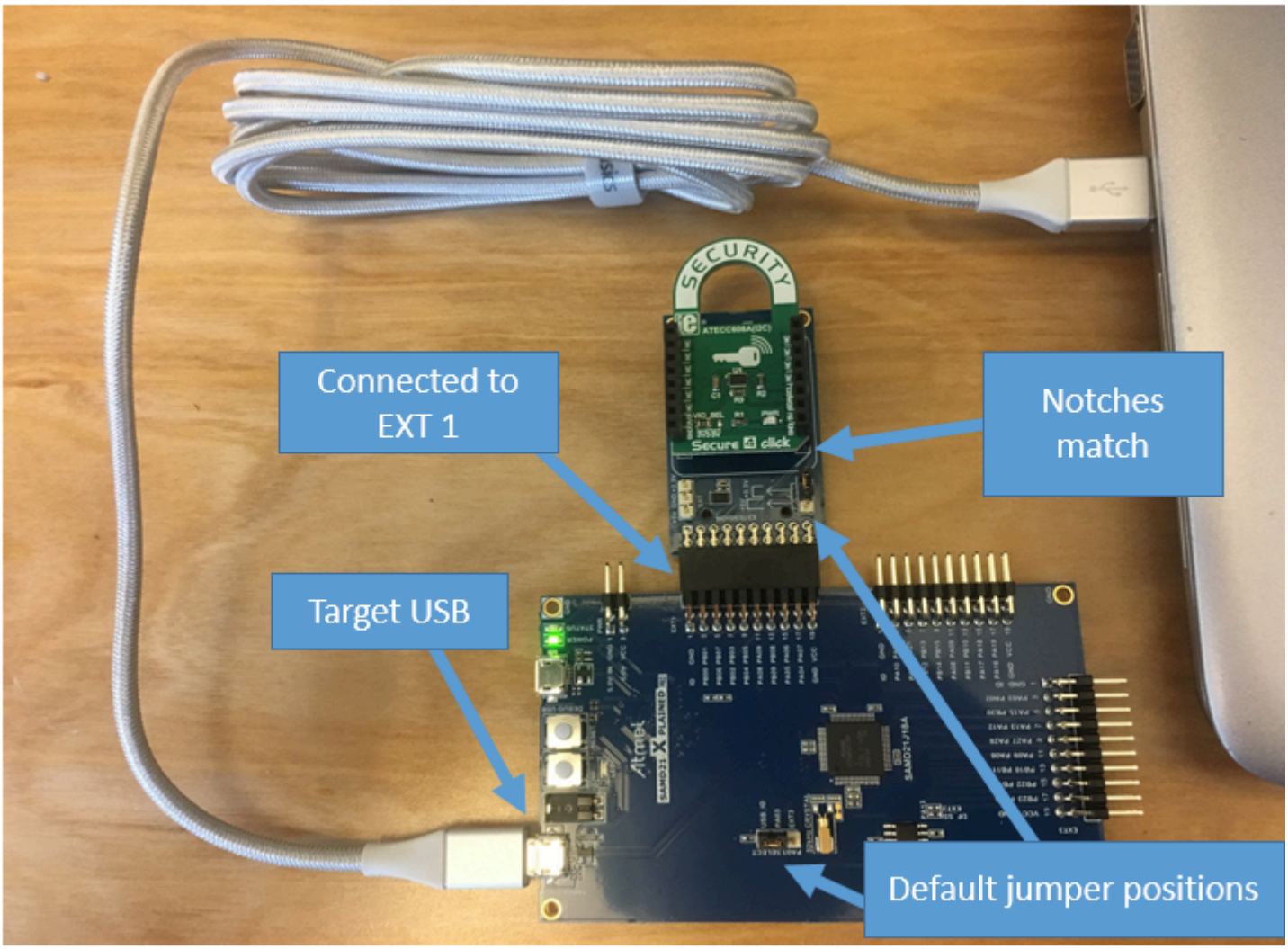
Para configurar la placa SAMD21 XPlained Pro

1. Siga el enlace CryptoAuthSSH-XSTK (DM320109): último firmware para descargar un archivo.zip que contiene instrucciones (PDF) y un binario que se puede programar en la D21.
2. Descargue e instale el IDP Atmel Studio 7. Asegúrese de seleccionar la arquitectura del controlador SMART ARM MCU durante la instalación.
3. Utilice un cable USB 2.0 Micro B para introducir el conector "Debug USB" al equipo y siga las instrucciones del PDF. (El conector "Debug USB" es el puerto USB más cercano al led POWER y los pines).

Para conectar el hardware

1. Desconecte el cable micro USB de Debug USB.
2. Conecte el adaptador mikroBUS XPlained Pro a la placa SAMD21 en la ubicación EXT1.
3. Conecte la placa ATECC608A Secure 4 Click al adaptador mikroBUSX XPlained Pro. Asegúrese de que la esquina dentada de la click board coincida con el icono dentado de la placa del adaptador.
4. Conecte el cable micro USB al USB de destino.

Su configuración debe ser similar a la siguiente imagen.



Configure el entorno de desarrollo.

Registro para obtener una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Cómo registrarse en 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.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación luego de completar 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.

Crear un usuario administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

Protección de su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Console como propietario de cuenta, elija Usuario raíz e ingrese el email de su Cuenta de AWS. 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 raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario administrativo

1. Activar IAM Identity Center

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

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

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

Cómo iniciar sesión como usuario administrativo

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

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

Para dar acceso, añada permisos a los usuarios, grupos o roles:

- Usuarios y grupos en AWS IAM Identity Center:

  Cree un conjunto de permisos. Siga las instrucciones de Creación de un conjunto de permisos en la Guía del usuario de AWS IAM Identity Center.

- Usuarios administrados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones de Creación de un rol para un proveedor de identidades de terceros (federación) en la Guía del usuario de IAM.

- Usuarios de IAM:
  - Cree un rol que el usuario pueda aceptar. Siga las instrucciones descritas en Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.
  - (No recomendado) Asocie una política directamente a un usuario o añada un usuario a un grupo de usuarios. Siga las instrucciones descritas en Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Configuración

1. Descargue el repositorio de FreeRTOS del repositorio de FreeRTOS. GitHub

   Para descargar Freertos desde: GitHub

   1. Navegue hasta el repositorio de Freertos GitHub .
   2. Elija Clone or download (Clonar o descargar).
   3. Desde la línea de comandos del equipo, clone el repositorio en un directorio de su equipo host.

      ```
      git clone https://github.com/aws/amazon-freertos.git -\-recurse-submodules
      ```

      Important

      - En este tema, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.
      - Los caracteres de espacio en la ruta *freertos* pueden causar errores de compilación. Al clonar o copiar el repositorio, asegúrese de que la ruta que crea no contiene caracteres de espacio.
      - La longitud máxima de una ruta de archivo en Microsoft Windows es de 260 caracteres. Las rutas largas al directorio de descargas de FreeRTOS pueden provocar errores de creación.
      - Como el código fuente puede contener enlaces simbólicos, si utiliza Windows para extraer el archivo, es posible que tenga que:
        - Habilitar el modo de desarrollador o
        - Utilizar una consola que tenga el rango de administrador.

        De esta forma, Windows puede crear correctamente enlaces simbólicos al extraer el archivo. De lo contrario, los enlaces simbólicos se escribirán como archivos normales que contengan las rutas de los enlaces simbólicos como texto o estarán vacíos. Para obtener más información, consulte la entrada del blog Symlinks in Windows 10 .

        Si usa Git en Windows, debe habilitar el modo desarrollador o debe:

        - Establecer `core.symlinks` en verdadero con el siguiente comando:

```
git config -\-global core.symlinks true
```

- Usar una consola que tenga el rango de administrador siempre que utilice un comando git que escriba en el sistema (por ejemplogit pull, git clone y git submodule update -\-init -\-recursive).

4. Desde el directorio *freertos* compruebe la rama que va a usar.

2. Configure el entorno de desarrollo.

   a. Instale la versión más reciente de WinPCap.

   b. Instale Microsoft Visual Studio.

      Se sabe que las versiones 2017 y 2019 de Visual Studio funcionan. Todas las ediciones de estas versiones de Visual Studio son compatibles (Community, Professional o Enterprise).

      Además del IDE, instale el componente Desarrollo de escritorio con C++. A continuación, en Optional (Opcional), instale el último SDK de Windows 10.

   c. Asegúrese de que tiene un activo conexión cableados.

Creación y ejecución del proyecto de demostración de FreeRTOS

> **Important**
>
> El microchip ATECC608A tiene una inicialización única que se bloquea en el dispositivo la primera vez que se ejecuta un proyecto (durante la llamada a `C_InitToken`). Sin embargo, el proyecto de prueba y el proyecto de demostración de FreeRTOS tienen diferentes configuraciones. Si el dispositivo está bloqueado durante las configuraciones del proyecto de demostración, no será posible que todas las pruebas del proyecto se realicen correctamente.

Para crear y ejecutar el proyecto de demostración de FreeRTOS con el IDE de Visual Studio

1. Cargue el proyecto en Visual Studio.

   En el menú File (Archivo), elija Open (Abrir). Elija File/Solution (Archivo/Solución), vaya al archivo *freertos*`\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln` y, a continuación, elija Open (Abrir).

2. Cambie el destino del proyecto de demostración.

   El proyecto de demostración depende del SDK de Windows, pero no tiene una versión de SDK de Windows especificada. De forma predeterminada, el IDE podría intentar compilar la demostración con una versión del SDK que no se encuentra en su equipo. Para establecer la versión del SDK de Windows, haga clic con el botón derecho del ratón en aws_demos y, a continuación, elija Retarget Projects (Redestinar proyectos). Se abrirá la ventana Revisar acciones de solución. Elija una versión del SDK de Windows que esté en su equipo (el valor inicial en el menú desplegable es suficiente) y, a continuación, elija OK (Aceptar).

3. Compile y ejecute el proyecto.

   En el menú Crear, elija Crear solución y asegúrese de que la solución se crea sin errores. Elija Debug, Start Debugging (Depurar, Iniciar depuración) para ejecutar el proyecto. En la primera ejecución, tendrá que configurar la interfaz del dispositivo y volver a compilar. Para obtener más información, consulte Configurar su interfaz de red.

4. Aprovisione el microchip ATECC608A.

   El microchip dispone de varias herramientas de scripting como ayuda para la configuración de los componentes de ATECC608A. Desplácese hasta *freertos*`\vendors\microchip\secure_elements\app\example_trust_chain_tool` y abra el archivo README.md.

   Siga las instrucciones del archivo `README.md` para aprovisionar el dispositivo. Estos pasos incluyen lo siguiente:

   1. Cree y registre una entidad de certificación en AWS.
   2. Genere sus claves en el microchip ATECC608A y exporte la clave pública y el número de serie del dispositivo.
   3. Genere un certificado para el dispositivo y regístrelo en AWS.
   4. Cargue el certificado de CA y el certificado de dispositivo en el dispositivo.

5. Cree y ejecute muestras de FreeRTOS.

   Vuelva a ejecutar el proyecto de demostración. Esta vez debería conectarse.

Solución de problemas

Para obtener información sobre la resolución de problemas, consulte Introducción a solución de problemas.

Cómo empezar con el Espressif ESP32- C y el ESP-WROVER-KIT DevKit

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

> **Note**
>
> Para explorar cómo integrar las bibliotecas modulares y demostraciones de FreeRTOS en su propio proyecto Espressif IDF, consulte nuestra integración de referencia destacada para la plataforma ESP32-C3.

Siga este tutorial para empezar a usar el Espressif ESP32- DevKit C equipado con los módulos ESP32-WROOM-32, ESP32-SOLO-1 o ESP-WROVER y el ESP-WROVER-KIT-VB. Para adquirir uno de nuestros socios en el Catálogo de dispositivos de socios de AWS, utilice los siguientes enlaces:

- EL ESP32-WROOM-32 C DevKit
- ESP32-SOLO-1
- ESP32-WROVER-KIT

Estas versiones de las placas de desarrollo con compatibles con FreeRTOS.

Para obtener más información sobre las versiones más recientes de estas placas, consulte la DevKitESP32-C V4 o la ESP-WROVER-KIT v4.1 en el sitio web de Espressif.

> **Note**
>
> Actualmente, el puerto Freertos para ESP32-WROVER-KIT y ESP DevKit C no admite la función de multiprocesamiento simétrico (SMP).

Información general

Este tutorial le guiará a través de los siguientes pasos:

1. Conexión de su placa a un equipo host.
2. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
3. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
4. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.
5. Interacción con la aplicación que se ejecuta en la placa con una conexión serie para fines de monitorización y depuración.

Requisitos previos

Antes de comenzar con FreeRTOS en la placa de Espressif, tiene que configurar la cuenta de AWS y los permisos.

Registro para obtener una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Cómo registrarse en 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.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación luego de completar 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.

Crear un usuario administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

Protección de su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Console como propietario de cuenta, elija Usuario raíz e ingrese el email de su Cuenta de AWS. 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 raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario administrativo

1. Activar IAM Identity Center

   Para conocer las instrucciones, consulte Habilitar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center.
2. En IAM Identity Center, otorga acceso administrativo a un usuario administrativo.

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

Cómo iniciar sesión como usuario administrativo

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

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

Para dar acceso, añada permisos a los usuarios, grupos o roles:

- Usuarios y grupos en AWS IAM Identity Center:

  Cree un conjunto de permisos. Siga las instrucciones de Creación de un conjunto de permisos en la Guía del usuario de AWS IAM Identity Center.

- Usuarios administrados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones de Creación de un rol para un proveedor de identidades de terceros (federación) en la Guía del usuario de IAM.

- Usuarios de IAM:
  - Cree un rol que el usuario pueda aceptar. Siga las instrucciones descritas en Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.
  - (No recomendado) Asocie una política directamente a un usuario o añada un usuario a un grupo de usuarios. Siga las instrucciones descritas en Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Introducción

> **Note**
>
> Los comandos de Linux de este tutorial requieren que utilice el intérprete de comandos Bash.

1. Configure el hardware de Espressif.
   - Para obtener información sobre cómo configurar el hardware de la placa de desarrollo ESP32-C, consulte la Guía de introducción al DevKit ESP32-C V4. DevKit
   - Para obtener información sobre cómo configurar el hardware de la placa de desarrollo ESP-WROVER-KIT, consulte Guía de introducción a ESP-WROVER-KIT V4.1.

   > **Important**
   >
   > Cuando llegue a la sección Introducción de las guías de Espressif, deténgase y vuelva a las instrucciones en esta página.

2. Descargue Amazon FreeRTOS desde. GitHub (Para ver las instrucciones, consulte el archivo README.md).

3. Configure el entorno de desarrollo.

   Debe instalar una cadena de herramientas para comunicarse con la placa. Espressif proporciona el ESP-IDF para desarrollar software para sus placas. Dado que el ESP-IDF tiene su propia versión del kernel de FreeRTOS integrada como componente, Amazon FreeRTOS incluye una versión personalizada del ESP-IDF v4.2 en la que se ha eliminado el kernel de FreeRTOS. Esto soluciona los problemas relacionados con los archivos duplicados al compilar. Para usar la versión personalizada del ESP-IDF v4.2 incluida en Amazon FreeRTOS, siga las instrucciones que aparecen a continuación para el sistema operativo de su máquina host.

   Windows

   1. Descargue el instalador en línea universal de ESP-IDF para Windows.
   2. Ejecute el instalador en línea universal.
   3. Cuando llegue al paso Descargar o usar ESP-IDF, seleccione Usar un directorio ESP-IDF existente y establezca Elegir directorio ESP-IDF existente en *freertos*`/vendors/espressif/esp-idf`.
   4. Complete la instalación.

   macOS

   1. Siga las instrucciones que se indican en Requisitos previos para la configuración estándar de la cadena de herramientas (ESP-IDF v4.2) para macOS.

      > **Important**
      >
      > Cuando llegue a las instrucciones de “Obtención de ESP-IDF” en Pasos siguientes, deténgase y vuelva a las instrucciones de esta página.

   2. Abra una ventana de línea de comandos.
   3. Navegue hasta el directorio de descargas de FreeRTOS y ejecute el siguiente script para descargar e instalar la cadena de herramientas espressif para su plataforma.

      ```
      vendors/espressif/esp-idf/install.sh
      ```

   4. Añada las herramientas de la cadena de herramientas ESP-IDF a la ruta de su terminal con el siguiente comando.

```
source vendors/espressif/esp-idf/export.sh
```

Linux

1. Siga las instrucciones que se indican en Requisitos previos para la configuración estándar de la cadena de herramientas (ESP-IDF v4.2) para Linux.

   > **Important**
   >
   > Cuando llegue a las instrucciones de “Obtención de ESP-IDF” en Pasos siguientes, deténgase y vuelva a las instrucciones de esta página.

2. Abra una ventana de línea de comandos.
3. Navegue hasta el directorio de descargas de FreeRTOS y ejecute el siguiente script para descargar e instalar la cadena de herramientas Espressif para su plataforma.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Añada las herramientas de la cadena de herramientas ESP-IDF a la ruta de su terminal con el siguiente comando.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

4. Establezca una conexión serie.

   a. Para establecer una conexión en serie entre su máquina host y el DevKit ESP32-C, debe instalar los controladores VCP USB a UART Bridge del CP210x. Puede descargar estos controladores de Silicon Labs.

      Para establecer una conexión entre su máquina host y el ESP32-WROVER-KIT, debe instalar el controlador de puerto COM virtual FTDI. Puede descargar este controlador desde FTDI.

   b. Siga los pasos para Establecer una conexión serie con ESP32.

   c. Después de establecer una conexión serie, anote el puerto serie de la conexión de la placa. Lo necesita para instalar la demostración.

Configuración de las aplicaciones de demostración de FreeRTOS

Para este tutorial, el archivo de configuración de FreeRTOS se encuentra en *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h. (Por ejemplo, si se elige AFR_BOARD espressif.esp32_devkitc, el archivo de configuración se encuentra en *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h).

1. Si ejecuta macOS o Linux, abra un símbolo del terminal. Si utiliza Windows, abra la aplicación “ESP-IDF 4.x CMD” (si incluyó esta opción al instalar la cadena de herramientas de ESP-IDF) o la aplicación de “símbolo del sistema” en caso contrario.
2. Para verificar que tiene instalado Python3, ejecute

   ```
   python --version
   ```

   Se muestra la versión instalada. Si no tiene instalado Python 3.0.1 o una versión posterior, puede instalarlo desde el sitio web de Python.
3. Necesita la interfaz de la línea de comandos (CLI) de AWS para ejecutar los comandos de AWS IoT. Si ejecuta Windows, utilice el comando easy_install awscli para instalar la CLI de AWS en la aplicación de “Comando” o “ESP-IDF 4.x CMD”.

   Si ejecuta macOS o Linux, consulte Instalación de la CLI de AWS.
4. Ejecute

   ```
   aws configure
   ```

   y configure la CLI de AWS con su ID de clave de acceso de AWS, la clave de acceso secreta y la región predeterminada de AWS. Para obtener más información, consulte Configuración de la CLI de AWS.
5. Utilice el comando siguiente para instalar el SDK de AWS para Python (boto3):

   - En Windows, en la aplicación de “Comando” o “ESP-IDF 4.x CMD”, ejecute

     ```
     pip install boto3 --user
     ```

> **Note**
>
> Consulte la Documentación de Boto3 para obtener más detalles.

- En macOS o Linux, ejecute

```
pip install tornado nose --user
```

y, a continuación, ejecute

```
pip install boto3 --user
```

FreeRTOS incluye el script `SetupAWS.py` para facilitar la configuración de su placa Espressif para conectar a AWS IoT. Para configurar el script, abra *`freertos`*`/tools/aws_config_quick_start/configure.json` y defina los siguientes atributos:

`afr_source_dir`

Ruta completa del directorio *`freertos`* de su equipo. Asegúrese de que utiliza barras diagonales para especificar esta ruta.

`thing_name`

El nombre que desea asignar al objeto de AWS IoT que representa la placa.

`wifi_ssid`

El SSID de su red wifi.

`wifi_password`

La contraseña para su red wifi.

`wifi_security`

El tipo de seguridad para su red wifi.

Los siguientes son tipos de seguridad válidos:

- `eWiFiSecurityOpen` (abierta, sin seguridad)
- `eWiFiSecurityWEP` (seguridad WEP)

- `eWiFiSecurityWPA` (seguridad WPA)
- `eWiFiSecurityWPA2` (seguridad WPA2)

6. Ejecute el script de configuración.

   a. Si ejecuta macOS o Linux, abra un símbolo del terminal. Si ejecuta Windows, abra la aplicación de “ESP-IDF 4.x CMD” o “Comando”.

   b. Vaya al directorio *`freertos`*`/tools/aws_config_quick_start` y ejecute

   ```
   python SetupAWS.py setup
   ```

   El script hace lo siguiente:

   - Crea un objeto de IoT, un certificado y una política.
   - Asocia la política de IoT al certificado y el certificado al objeto de AWS IoT.
   - Rellena el archivo `aws_clientcredential.h` con su punto de conexión de AWS IoT, el SSID de la red wifi y las credenciales.
   - Formatea el certificado y la clave privada y los escribe en el archivo de encabezado `aws_clientcredential_keys.h`.

   > **Note**
   >
   > El certificado tiene codificación fija únicamente con fines ilustrativos. Las aplicaciones de producción deben almacenar estos archivos en un lugar seguro.

   Para obtener más información sobre `SetupAWS.py`, consulte el archivo `README.md` en el directorio *`freertos`*`/tools/aws_config_quick_start`.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Vaya a la consola de AWS IoT.

2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT.
3. En Tema de suscripción, escriba *your-thing-name*`/example/topic` y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje "¡Hola, mundo!" enviado varias veces al tema al que se ha suscrito.

Creación, instalación y ejecución del proyecto de demostración de FreeRTOS con el script idf.py

Puede usar la utilidad de IDF de Espressif (`idf.py`) para crear el proyecto e instalar los archivos binarios en su dispositivo.

> **Note**
>
> Algunas configuraciones pueden requerir que utilice la opción de puerto `"-p port-name"` con `idf.py` para especificar el puerto correcto, como en el siguiente ejemplo.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Creación e instalación de FreeRTOS en Windows, Linux y macOS (ESP-IDF v4.2)

1. Desplácese hasta la raíz del directorio de descargas de FreeRTOS.
2. En una ventana de línea de comandos, introduzca el siguiente comando para añadir las herramientas ESP-IDF a la ruta del terminal.

   Windows (aplicación "Comando")

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (aplicación "ESP-IDF 4.x CMD")

   (Esto ya se hizo cuando abrió aplicación).

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. Configure cmake en el directorio `build` y cree la imagen del firmware con el siguiente comando.

```
idf.py -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 build
```

Debería ver un resultado como el siguiente.

```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
   Warn about uninitialized values.
   -- Found Git: /usr/bin/git (found version "2.17.0")
   -- Building empty aws_iot component due to configuration
   -- Component names: ...
   -- Component paths: ...

   ... (more lines of build system output)

   [527/527] Generating hello-world.bin
   esptool.py v2.3.1

   Project build complete. To flash, run this command:
   ../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600
 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
hello-world.bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/
partition_table/partition-table.bin
   or run 'idf.py -p PORT flash'
```

Si no hay errores, la creación generará los archivos .bin binarios del firmware.

4. Borre la memoria flash de la placa de desarrollo con el siguiente comando.

```
idf.py erase_flash
```

5. Utilice el script `idf.py` para instalar el archivo binario de la aplicación en la placa.

```
idf.py flash
```

6. Supervise la salida del puerto serie de la placa con el siguiente comando.

```
idf.py monitor
```

> **Note**
>
> Puede combinar estos comandos como se muestra en el siguiente ejemplo.
>
> ```
> idf.py erase_flash flash monitor
> ```

Para determinadas configuraciones de máquinas host, debe especificar el puerto en el que se va a instalar la placa, como en el siguiente ejemplo.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

Creación e instalación de FreeRTOS con CMake

Además de usar el script `idf.py` proporcionado por el SDK de IDF para crear y ejecutar su código, también puede crear el proyecto con CMake. Actualmente es compatible con Unix Makefiles y el sistema de creación Ninja.

Para crear e instalar el proyecto

1. En una ventana de línea de comandos, navegue hasta el directorio de descargas de FreeRTOS.
2. Ejecute el siguiente script para añadir las herramientas ESP-IDF a la ruta de su intérprete de comandos.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Utilice el siguiente comando para generar los archivos de creación.

Con Unix Makefiles

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
```

Con Ninja

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
```

4. Compilar el proyecto.

   Con Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

   Con Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY -j8
   ```

5. Borre la instalación y, a continuación, instale la placa.

   Con Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Con Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Ejecución de las demostraciones de Bluetooth de bajo consumo

Freertos admite la conectividad Biblioteca de Bluetooth de bajo consumo.

Para ejecutar el proyecto de demostración de FreeRTOS en Bluetooth de bajo consumo, debe ejecutar la aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS en su dispositivo móvil iOS o Android.

Para configurar la aplicación de demostración del SDK para móviles de Bluetooth de bajo consumo de FreeRTOS

1. Siga las instrucciones de SDK para móviles para dispositivos Bluetooth de FreeRTOS para descargar e instalar los SDK para su plataforma móvil en su equipo host.
2. Siga las instrucciones en Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS para configurar la aplicación móvil de demostración en su dispositivo móvil.

Para obtener instrucciones sobre cómo ejecutar la demostración de Bluetooth de bajo consumo de MQTT en la placa, consulte MQTT a través de Bluetooth de bajo consumo.

Para obtener instrucciones sobre cómo ejecutar la demostración de aprovisionamiento wifi en la placa, consulte Aprovisionamiento Wi-Fi.

Uso de FreeRTOS en su propio proyecto de CMake para ESP32

Si desea usar FreeRTOS en su propio proyecto de CMake, puede configurarlo como un subdirectorio y crearlo junto con su aplicación. En primer lugar, obtenga una copia de FreeRTOS en. GitHub También puede configurarlo como un submódulo git con el siguiente comando para que sea más fácil actualizarlo en el futuro.

```
git submodule add -b release https://github.com/aws/amazon-freertos.git freertos
```

Si se publica una versión más reciente, puede actualizar su copia local con estos comandos.

```
# Pull the latest changes from the remote tracking branch.
git submodule update --remote -- freertos
```

```
# Commit the submodule change because it is pointing to a different revision now.
git add freertos
```

```
git commit -m "Update FreeRTOS to a new release"
```

Si el proyecto tiene la siguiente estructura de directorios:

```
- freertos (the copy that you obtained from GitHub or the AWS IoT console)
- src
  - main.c (your application code)
- CMakeLists.txt
```

A continuación, se muestra un ejemplo del archivo `CMakeLists.txt` de nivel superior que se puede utilizar para crear la aplicación junto con FreeRTOS.

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Tell IDF build to link against this target.
set(IDF_EXECUTABLE_SRCS "<complete_path>/src/main.c")
set(IDF_PROJECT_EXECUTABLE my_app)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

Para generar el proyecto, ejecute los siguientes comandos de CMake. Asegúrese de que el compilador ESP32 está en la variable de entorno PATH.

```
cmake -S . -B build-directory -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/
xtensa-esp32.cmake -GNinja
```

```
cmake --build build-directory
```

Para instalar la aplicación en su placa, ejecute el siguiente comando.

```
cmake --build build-directory --target flash
```

Uso de los componentes de FreeRTOS

Después de ejecutar CMake, puede encontrar todos los componentes disponibles en la salida de resumen. Debería parecerse al siguiente ejemplo.

```
====================Configuration for FreeRTOS====================
  Version:                 202107.00
  Git version:             202107.00-g79ad6defb

Target microcontroller:
  vendor:                  Espressif
  board:                   ESP32-DevKitC
  description:             Development board produced by Espressif that comes in two
                           variants either with ESP-WROOM-32 or ESP32-WROVER module
  family:                  ESP32
  data ram size:           520KB
  program memory size:     4MB

Host platform:
  OS:                      Linux-4.15.0-66-generic
  Toolchain:               xtensa-esp32
  Toolchain path:          /opt/xtensa-esp32-elf
  CMake generator:         Ninja

FreeRTOS modules:
  Modules to build:        backoff_algorithm, common, common_io, core_http,
                           core_http_demo_dependencies, core_json, core_mqtt,
                           core_mqtt_agent, core_mqtt_agent_demo_dependencies,
                           core_mqtt_demo_dependencies, crypto, defender, dev_mode_key_
                           provisioning, device_defender, device_defender_demo_
                           dependencies, device_shadow,
 device_shadow_demo_dependencies,
                           freertos_cli_plus_uart, freertos_plus_cli, greengrass,
                           http_demo_helpers, https, jobs, jobs_demo_dependencies,
                           kernel, logging, mqtt, mqtt_agent_interface, mqtt_demo_
                           helpers, mqtt_subscription_manager, ota, ota_demo_
                           dependencies, ota_demo_version, pkcs11, pkcs11_helpers,
                           pkcs11_implementation, pkcs11_utils, platform,
 secure_sockets,
                           serializer, shadow, tls, transport_interface_secure_sockets,
                           wifi
  Enabled by user:         common_io, core_http_demo_dependencies, core_json,
                           core_mqtt_agent_demo_dependencies, core_mqtt_demo_
```

```
                            dependencies, defender, device_defender,
 device_defender_demo_
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
 https,
                            jobs, jobs_demo_dependencies, logging,
 ota_demo_dependencies,
                            pkcs11, pkcs11_helpers, pkcs11_implementation, pkcs11_utils,
                            platform, secure_sockets, shadow, wifi
  Enabled by dependency:    backoff_algorithm, common, core_http, core_mqtt,
                            core_mqtt_agent, crypto, demo_base,
 dev_mode_key_provisioning,
                            freertos, http_demo_helpers, kernel, mqtt, mqtt_agent_
                            interface, mqtt_demo_helpers, mqtt_subscription_manager,
 ota,
                            ota_demo_version, pkcs11_mbedtls, serializer, tls,
                            transport_interface_secure_sockets, utils
  3rdparty dependencies:    jsmn, mbedtls, pkcs11, tinycbor
  Available demos:          demo_cli_uart, demo_core_http, demo_core_mqtt,
 demo_core_mqtt_
                            agent, demo_device_defender, demo_device_shadow,
                            demo_greengrass_connectivity, demo_jobs, demo_ota_core_http,
                            demo_ota_core_mqtt, demo_tcp
  Available tests:
=======================================================================
```

Puede hacer referencia a cualquier componente de la lista `Modules to build`. Para vincularlos a su aplicación, coloque el espacio de nombres `AFR::` delante del nombre (por ejemplo `AFR::core_mqtt`, `AFR::ota`, etc.).

Adición de componentes personalizados a ESP-IDF

Puede añadir más componentes mientras utiliza ESP-IDF. Por ejemplo, suponiendo que desea añadir un componente llamado `example_component` y su proyecto tiene el siguiente aspecto:

```
- freertos
- components
  - example_component
    - include
      - example_component.h
    - src
      - example_component.c
```

```
    - CMakeLists.txt
- src
  - main.c
- CMakeLists.txt
```

A continuación, se muestra un ejemplo del archivo `CMakeLists.txt` de su componente.

```
add_library(example_component src/example_component.c)
target_include_directories(example_component PUBLIC include)
```

A continuación, en el archivo `CMakeLists.txt` de nivel superior, añada el componente insertando la siguiente línea justo después de `add_subdirectory(freertos)`.

```
add_subdirectory(component/example_component)
```

A continuación, modifique `target_link_libraries` para incluir su componente.

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)
```

Este componente ahora se vincula automáticamente al código de la aplicación de forma predeterminada. Ahora puede incluir sus archivos de encabezado y llamar a las funciones que define.

Anulación de las configuraciones para FreeRTOS

Actualmente no hay un enfoque bien definido para redefinir las configuraciones fuera del árbol de código fuente de FreeRTOS. De forma predeterminada, CMake buscará los directorios *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ y *freertos*/demos/include/. Sin embargo, puede usar una solución alternativa para indicar al compilador que busque primero en otros directorios. Por ejemplo, puede añadir otra carpeta para las configuraciones de FreeRTOS.

```
- freertos
- freertos-configs
  - aws_clientcredential.h
  - aws_clientcredential_keys.h
  - iot_mqtt_agent_config.h
  - iot_config.h
```

```
- components
- src
- CMakeLists.txt
```

Los archivos de `freertos-configs` se copian de los directorios *`freertos`*`/vendors/espressif/boards/esp32/aws_demos/config_files/` y *`freertos`*`/demos/include/`. A continuación, en el archivo de nivel superior `CMakeLists.txt`, agregue esta línea delante de `add_subdirectory(freertos)` para que el compilador busque primero en este directorio.

```
include_directories(BEFORE freertos-configs)
```

Proporcionar su propio sdkconfig para ESP-IDF

En caso de que desee proporcionar su propio `sdkconfig.default`, puede establecer la variable de CMake `IDF_SDKCONFIG_DEFAULTS` desde la línea de comandos:

```
cmake -S . -B build-directory -DIDF_SDKCONFIG_DEFAULTS=path_to_your_sdkconfig_defaults
 -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/xtensa-esp32.cmake -GNinja
```

Si no especifica una ubicación para su propio archivo `sdkconfig.default`, FreeRTOS utilizará el archivo predeterminado ubicado en *`freertos`*`/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults`.

Para obtener más información, consulte Configuración del proyecto en la Referencia de la API de Espressif y, si encuentra problemas después de haber compilado correctamente, consulte la sección sobre Opciones obsoletas y sus sustituciones en esa página.

Resumen

Si tiene un proyecto con un componente llamado `example_component` y desea invalidar algunas configuraciones, aquí tiene un ejemplo completo del archivo `CMakeLists.txt` de nivel superior.

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

set(IDF_PROJECT_EXECUTABLE my_app)
set(IDF_EXECUTABLE_SRCS "src/main.c")
```

```
# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)

# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is a variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/example_component" ABSOLUTE
)
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})

# Override the configurations for FreeRTOS.
include_directories(BEFORE freertos-configs)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

Solución de problemas

- Si ejecuta macOS y el sistema operativo no reconoce su ESP-WROVER-KIT, asegúrese de que no tiene los controladores D2XX instalados. Para desinstalarlos, siga las instrucciones de la guía de instalación de controladores FTDI para macOS X.
- La utilidad de monitorización proporcionada por ESP-IDF (e invocada mediante la supervisión de creación) le ayuda a descodificar las direcciones. Por este motivo, puede ayudarle a obtener algunos rastreos inversos significativos en caso de que se produzca el bloqueo de la aplicación. Para obtener más información, consulte Descodificación automática de direcciones en el sitio web de Espressif.
- También es posible habilitar GDBstub para la comunicación con gdb sin necesidad de hardware JTAG especial. Para obtener más información, consulte Inicio de GDB para GDBStub en el sitio web de Espressif.
- Para obtener información acerca de cómo configurar un entorno basado en OpenOCD si fuera necesario realizar la depuración basada en hardware JTAG, consulte Depuración de JTAG en el sitio web de Espressif.
- Si no es posible instalar `pyserial` utilizando `pip` en macOS, descárguelo del sitio web de pyserial.

- Si la placa se restablece de forma continua, intente borrar la instalación escribiendo el siguiente comando en el terminal.

```
make erase_flash
```

- Si ve errores al ejecutar `idf_monitor.py`, utilice Python 2.7.
- Las bibliotecas necesarias de ESP-IDF se incluyen en FreeRTOS, por lo que no es necesario descargarlas de forma externa. Si se ha establecido la variable del entorno `IDF_PATH`, le recomendamos que la elimine antes de la creación de FreeRTOS.
- En Windows, el proyecto puede tardar entre 3 y 4 minutos en compilarse. Para reducir el tiempo de creación, puede utilizar el conmutador `-j4` del comando make.

```
make flash monitor -j4
```

- Si el dispositivo tiene dificultades para conectarse a AWS IoT, abra el archivo `aws_clientcredential.h` y compruebe que las variables de configuración estén correctamente definidas en el archivo. `clientcredentialMQTT_BROKER_ENDPOINT[]` debe tener un aspecto similar al siguiente: `1234567890123-ats.iot.us-east-1.amazonaws.com`.
- Si sigue los pasos de Uso de FreeRTOS en su propio proyecto de CMake para ESP32 y ve errores de referencias no definidas del vinculador, generalmente se debe a que faltan bibliotecas o demostraciones dependientes. Para agregarlas, actualice el archivo `CMakeLists.txt` (en el directorio raíz) usando la función CMake estándar `target_link_libraries`.
- La versión 4.2 de ESP-IDF admite el uso de la cadena de herramientas xtensa\-esp32\-elf\-gcc 8\.2\.0\.. Si utiliza una versión anterior de la cadena de herramientas de Xtensa, descargue la versión necesaria.
- Si ve un registro de errores como el siguiente sobre las dependencias de Python que no se cumplen en la versión 4.2 de ESP-IDF:

```
The following Python requirements are not satisfied:
  click>=5.0
  pyserial>=3.0
  future>=0.15.2
  pyparsing>=2.0.3,<2.4.0
  pyelftools>=0.22
  gdbgui==0.13.2.0
  pygdbmi<=0.9.0.2
  reedsolo>=1.5.3,<=1.5.4
  bitstring>=3.1.6
```

```
 ecdsa>=0.16.0
 Please follow the instructions found in the "Set up the tools" section of ESP-IDF
Getting Started Guide
```

Instale las dependencias de Python en su plataforma mediante el siguiente comando de Python:

```
root/vendors/espressif/esp-idf/requirements.txt
```

Para obtener más información sobre cómo solucionar problemas, consulte Introducción a solución de problemas.

Debugging

Código de depuración en Espressif ESP32- DevKit C y ESP-WROVER-KIT (ESP-IDF v4.2)

En esta sección se muestra cómo utilizar la versión 4.2 de ESP-IDF en el hardware de Espressif. Necesita un cable que vaya de JTAG al USB. Utilizamos un cable de USB a MPSSE (por ejemplo, el C232HM-DDHSL-0 de FTDI).

Configuración DevKit del ESP-C JTAG

Para el cable C232HM-DDHSL-0 de FTDI, se trata de las conexiones al ESP32 DevkitC.

```
| C232HM-DDHSL-0 Wire Color | ESP32 GPIO Pin | JTAG Signal Name |
| ------------------------- | -------------- | ---------------- |
|  Brown (pin 5)            |  IO14          |  TMS             |
|  Yellow (pin 3)           |  IO12          |  TDI             |
|  Black (pin 10)           |  GND           |  GND             |
|  Orange (pin 2)           |  IO13          |  TCK             |
|  Green (pin 4)            |  IO15          |  TDO             |
```

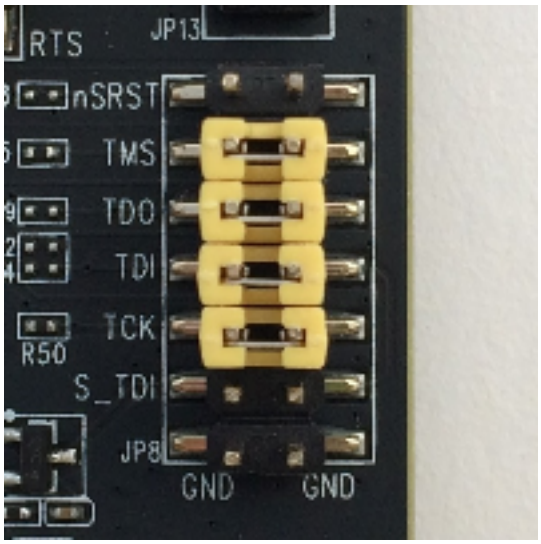
Configuración de JTAG de ESP-WROVER-KIT

Para el cable C232HM-DDHSL-0 de FTDI, se trata de las conexiones al ESP32-WROVER-KIT.

```
| C232HM-DDHSL-0 Wire Color | ESP32 GPIO Pin | JTAG Signal Name |
| ------------------------- | -------------- | ---------------- |
|  Brown (pin 5)            |  IO14          |  TMS             |
|  Yellow (pin 3)           |  IO12          |  TDI             |
|  Orange (pin 2)           |  IO13          |  TCK             |
|  Green (pin 4)            |  IO15          |  TDO             |
```

Estas tablas se han desarrollado a partir de lahoja de datos de C232HM-DDHSL-0 de FTDI. Para obtener más información, consulte la sección Conexión del cable C232HM MPSSE y detalles mecánicos en la hoja de datos.

Para habilitar JTAG en el ESP-WROVER-KIT, coloque puentes en los pines TMS, TDO, TDI, TCK y S_TDI tal y como se muestra aquí.



Depuración en Windows (ESP-IDF v4.2)

Para configurar para la depuración en Windows

1. Conecte el lado del USB de C232HM-DDHSL-0 de FTDI en su equipo y el otro lado tal y como se describe en Código de depuración en Espressif ESP32- DevKit C y ESP-WROVER-KIT (ESP-IDF v4.2). El dispositivo C232HM-DDHSL-0 de FTDI debe aparecer en Device Manager (Administrador de dispositivos) bajo Universal Serial Bus Controllers (Controladores de bus serie universales).
2. En la lista de dispositivos de bus serie universal, haga clic con el botón derecho en el dispositivo C232HM-DDHSL-0 y elija Propiedades.

> **Note**
>
> El dispositivo podría aparecer como USB Serial Port (Puerto serie USB).

En la ventana de propiedades, elija la pestaña Detalles para ver las propiedades del dispositivo. Si el dispositivo no aparece en la lista, instale el controlador de Windows para C232HM-DDHSL-0 de FTDI.

3. En la pestaña Details (Detalles), elija Property (Propiedad) y, a continuación, elija Hardware IDs (ID de hardware). Debería ver algo similar a lo siguiente en el campo Valor.

   ```
   FTDIBUS\COMPORT&VID_0403&PID_6014
   ```

   En este ejemplo, el ID de proveedor es 0403 y el ID de producto es 6014.

   Compruebe que estos ID coinciden con los ID de `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Los ID se especifican en una línea que comienza con `ftdi_vid_pid` seguida de un ID de proveedor y un ID de producto.

   ```
   ftdi_vid_pid 0x0403 0x6014
   ```

4. Descargue OpenOCD para Windows.
5. Descomprima el archivo en `C:\` y añada `C:\openocd-esp32\bin` a la ruta del sistema.
6. OpenOCD requiere libusb, que no está instalado de forma predeterminada en Windows. Para instalar libusb:

   a. Descargue zadig.exe.

   b. Ejecute `zadig.exe`. Desde el menú Options (Opciones), seleccione List All Devices (Lista de todos los dispositivos).

   c. En el menú desplegable, elija C232HM-DDHSL-0.

   d. En el campo del controlador de destino, que se encuentra a la derecha de la flecha verde, elija WinUSB.

   e. En la lista que hay bajo el campo del controlador de destino, elija la flecha y, a continuación, haga elija Instalar controlador. Elija Replace Driver (Reemplazar controlador).

7. Abra un símbolo del sistema, vaya a la raíz del directorio de descargas de FreeRTOS y ejecute el siguiente comando.

   ```
   idf.py openocd
   ```

   Deje este símbolo del sistema abierto.

8. Abra un nuevo símbolo del sistema, vaya a la raíz del directorio de descargas de FreeRTOS y ejecute

```
idf.py flash monitor
```

9. Abra otra línea de comandos, navegue hasta la raíz de su directorio de descargas de FreeRTOS y espere a que la demostración comience a ejecutarse en su placa. Cuando esto suceda, ejecute

   ```
   idf.py gdb
   ```

   El programa debe detenerse en la función `main`.

   > **Note**
   >
   > El ESP32 admite un máximo de dos puntos de interrupción.

Depuración en macOS (ESP-IDF v4.2)

1. Descargue el controlador FTDI para macOS.
2. Descargue OpenOCD.
3. Extraiga el archivo.tar descargado y establezca la ruta de `.bash_profile` en `OCD_INSTALL_DIR/openocd-esp32/bin`.
4. Utilice el siguiente comando para instalar `libusb` en macOS.

   ```
   brew install libusb
   ```

5. Utilice el siguiente comando para descargar el controlador de puerto serie.

   ```
   sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
   ```

6. Utilice el siguiente comando para descargar el controlador de puerto serie.

   ```
   sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
   ```

7. Si ejecuta una versión de macOS posterior a 10.9, utilice el siguiente comando para descargar el controlador FTDI de Apple.

   ```
   sudo kextunload -b com.apple.driver.AppleUSBFTDI
   ```

8. Utilice el siguiente comando para obtener el ID del producto y el ID de proveedor ID del cable FTDI. Enumera los dispositivos USB conectados.

   ```
   system_profiler SPUSBDataType
   ```

   La salida de `system_profiler` debería ser similar a la siguiente.

   ```
   DEVICE:

   Product ID: product-ID
   Vendor ID: vendor-ID (Future Technology Devices International Limited)
   ```

9. Abra el archivo `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. El ID del proveedor y el ID del producto de su dispositivo se especifican en una línea que comienza por `ftdi_vid_pid`. Cambie los ID para que coincidan con los del resultado de `system_profiler` del paso anterior.
10. Abra una ventana de terminal, navegue hasta la raíz de su directorio de descargas de FreeRTOS y utilice el comando siguiente para ejecutar OpenOCD.

    ```
    idf.py openocd
    ```

    Deje abierta esta ventana de terminal.

11. Abra un nuevo terminal y utilice el siguiente comando para cargar el controlador de puerto serie FTDI.

    ```
    sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
    ```

12. Desplácese hasta la raíz del directorio de descargas de FreeRTOS y ejecute

    ```
    idf.py flash monitor
    ```

13. Abra un nuevo terminal, vaya a la raíz del directorio de descargas de FreeRTOS y ejecute

    ```
    idf.py gdb
    ```

    El programa debe detenerse en `main`.

Depuración en Linux (ESP-IDF v4.2)

1. Descargue OpenOCD. Extraiga el archivo tarball y siga las instrucciones de instalación en el archivo readme.
2. Para instalar libusb en Linux, use el siguiente comando.

```
sudo apt-get install libusb-1.0
```

3. Abra un terminal e introduzca USB ls -l /dev/ttyUSB* para crear una lista de todos los dispositivos conectados a su equipo. Esto le ayuda a comprobar si el sistema operativo reconoce los puertos USB de la placa. Debería ver un resultado como el siguiente.

```
   $ls -l /dev/ttyUSB*
   crw-rw----    1    root    dialout    188,    0    Jul    10    19:04    /
dev/ttyUSB0
   crw-rw----    1    root    dialout    188,    1    Jul    10    19:04    /
dev/ttyUSB1
```

4. Cierre la sesión y, a continuación, inicie sesión y realice un ciclo de encendido y apagado para la placa para que los cambios surtan efecto. En un símbolo del terminal, enumere los dispositivos USB. Asegúrese de que el responsable del grupo ha cambiado de `dialout` a `plugdev`.

```
   $ls -l /dev/ttyUSB*
   crw-rw----    1    root    plugdev    188,    0    Jul    10    19:04    /
dev/ttyUSB0
   crw-rw----    1    root    plugdev    188,    1    Jul    10    19:04    /
dev/ttyUSB1
```

   La interfaz `/dev/ttyUSBn` con el número más bajo se utiliza para la comunicación JTAG. La otra interfaz se dirige al puerto serie de ESP32 (UART) y se utiliza para cargar el código en la memoria flash del ESP32.

5. En una ventana de terminal, navegue hasta la raíz de su directorio de descargas de FreeRTOS y utilice el comando siguiente para ejecutar OpenOCD.

```
idf.py openocd
```

6. Abra otro terminal, vaya a la raíz del directorio de descargas de FreeRTOS y ejecute el siguiente comando.

```
idf.py flash monitor
```

7. Abra otro terminal, vaya a la raíz del directorio de descargas de FreeRTOS y ejecute el siguiente comando:

   ```
   idf.py gdb
   ```

   El programa debe detenerse en `main()`.

Introducción a Espressif ESP32-WROOM-32SE

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

> **Note**
>
> - Para explorar cómo integrar las bibliotecas modulares y demostraciones de FreeRTOS en su propio proyecto Espressif IDF, consulte nuestra integración de referencia destacada para la plataforma ESP32-C3.
> - Actualmente, el puerto Freertos para ESP32-WROOM-32SE no admite la función de multiprocesamiento simétrico (SMP).

Este tutorial muestra cómo empezar a utilizar Espressif ESP32-WROOM-32SE. Para adquirir uno de nuestros socios en el Catálogo de dispositivos de socios de AWS, consulte ESP32-WROOM-32SE.

Información general

Este tutorial le guiará a través de los siguientes pasos:

1. Conecte la placa a un equipo host.

2. Instale el software en su equipo host para desarrollar y depurar las aplicaciones integradas de la placa del microcontrolador.
3. Realice la compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
4. Cargue la imagen binaria de la aplicación en su placa y, a continuación, ejecute la aplicación.
5. Monitorice y depure la aplicación en ejecución mediante una conexión serie.

Requisitos previos

Antes de comenzar con FreeRTOS en la placa de Espressif, tiene que configurar la cuenta de AWS y los permisos.

Registro para obtener una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Cómo registrarse en 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.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación luego de completar 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.

Crear un usuario administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

Protección de su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Console como propietario de cuenta, elija Usuario raíz e ingrese el email de su Cuenta de AWS. 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 raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario administrativo

1. Activar IAM Identity Center

   Para conocer las instrucciones, consulte Habilitar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center.
2. En IAM Identity Center, otorga acceso administrativo a un usuario administrativo.

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

Cómo iniciar sesión como usuario administrativo

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

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

Para dar acceso, añada permisos a los usuarios, grupos o roles:

- Usuarios y grupos en AWS IAM Identity Center:

  Cree un conjunto de permisos. Siga las instrucciones de Creación de un conjunto de permisos en la Guía del usuario de AWS IAM Identity Center.

- Usuarios administrados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones de Creación de un rol para un proveedor de identidades de terceros (federación) en la Guía del usuario de IAM.

- Usuarios de IAM:
  - Cree un rol que el usuario pueda aceptar. Siga las instrucciones descritas en Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.
  - (No recomendado) Asocie una política directamente a un usuario o añada un usuario a un grupo de usuarios. Siga las instrucciones descritas en Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Introducción

> **Note**
>
> Los comandos de Linux de este tutorial requieren que utilice el intérprete de comandos Bash.

1. Configure el hardware de Espressif.

   Para obtener información sobre la configuración del hardware de la placa de desarrollo ESP32-WROOM-32SE, consulte la Guía de introducción al ESP32-C V4. DevKit

   > **Important**
   >
   > Cuando llegue a la sección Instalación paso a paso de la guía, siga hasta completar el paso 4 (Configuración de las variables de entorno). Deténgase después de completar el paso 4 y siga los pasos restantes aquí.

2. Descargue Amazon FreeRTOS desde. GitHub (Para ver las instrucciones, consulte el archivo README.md).
3. Configure el entorno de desarrollo.

   Debe instalar una cadena de herramientas para comunicarse con la placa. Espressif proporciona el ESP-IDF para desarrollar software para sus placas. Dado que el ESP-IDF tiene su propia versión del kernel de FreeRTOS integrada como componente, Amazon FreeRTOS incluye una versión personalizada del ESP-IDF v4.2 en la que se ha eliminado el kernel de FreeRTOS. Esto

soluciona los problemas relacionados con los archivos duplicados al compilar. Para usar la versión personalizada del ESP-IDF v4.2 incluida en Amazon FreeRTOS, siga las instrucciones que aparecen a continuación para el sistema operativo de su máquina host.

Windows

1. Descargue el instalador en línea universal de ESP-IDF para Windows.
2. Ejecute el instalador en línea universal.
3. Cuando llegue al paso Descargar o usar ESP-IDF, seleccione Usar un directorio ESP-IDF existente y establezca Elegir directorio ESP-IDF existente en *freertos*/vendors/espressif/esp-idf.
4. Complete la instalación.

macOS

1. Siga las instrucciones que se indican en Requisitos previos para la configuración estándar de la cadena de herramientas (ESP-IDF v4.2) para macOS.

   > **Important**
   >
   > Cuando llegue a las instrucciones de “Obtención de ESP-IDF” en Pasos siguientes, deténgase y vuelva a las instrucciones de esta página.

2. Abra una ventana de línea de comandos.
3. Navegue hasta el directorio de descargas de FreeRTOS y ejecute el siguiente script para descargar e instalar la cadena de herramientas espressif para su plataforma.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Añada las herramientas de la cadena de herramientas ESP-IDF a la ruta de su terminal con el siguiente comando.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

Linux

1. Siga las instrucciones que se indican en Requisitos previos para la configuración estándar de la cadena de herramientas (ESP-IDF v4.2) para Linux.

   > **Important**
   >
   > Cuando llegue a las instrucciones de “Obtención de ESP-IDF” en Pasos siguientes, deténgase y vuelva a las instrucciones de esta página.

2. Abra una ventana de línea de comandos.
3. Navegue hasta el directorio de descargas de FreeRTOS y ejecute el siguiente script para descargar e instalar la cadena de herramientas Espressif para su plataforma.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Añada las herramientas de la cadena de herramientas ESP-IDF a la ruta de su terminal con el siguiente comando.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

4. Establezca una conexión serie.

   a. Para establecer una conexión entre su máquina host y el ESP32-WROOM-32SE, instale los controladores de VCP de puente del CP210x USB a UART. Puede descargar estos controladores de Silicon Labs.

   b. Siga los pasos para establecer una conexión serie con ESP32.

   c. Después de establecer una conexión serie, anote el puerto serie de la conexión de la placa. Lo necesita para instalar la demostración.

Configuración de las aplicaciones de demostración de FreeRTOS

Para este tutorial, el archivo de configuración de FreeRTOS se encuentra en *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h. (Por ejemplo, si se elige `AFR_BOARD espressif.esp32_devkitc`, el archivo de configuración se encuentra en *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h).

> **Important**
>
> El dispositivo ATECC608A tiene una inicialización única que se bloquea en el dispositivo la primera vez que se ejecuta un proyecto (durante la llamada a `C_InitToken`). Sin embargo, el proyecto de prueba y el proyecto de demostración de FreeRTOS tienen diferentes configuraciones. Si el dispositivo está bloqueado durante las configuraciones del proyecto de demostración, no todas las pruebas del proyecto de prueba tendrán éxito.

1. Configure el proyecto de demostración de FreeRTOS siguiendo los pasos que se describen en Configuración de las demostraciones de FreeRTOS. Cuando llegue al último paso Para formatear sus credenciales de AWS IoT, deténgase y realice los siguientes pasos.
2. El microchip dispone de varias herramientas de scripting como ayuda para la configuración de los componentes de ATECC608A. Vaya al directorio *`freertos`*`/vendors/microchip/example_trust_chain_tool` y abra el archivo `README.md`.
3. Siga las instrucciones del archivo `README.md` para aprovisionar el dispositivo. Estos pasos incluyen lo siguiente:
    1. Cree y registre una entidad de certificación en AWS.
    2. Genere sus claves en el ATECC608A y exporte la clave pública y el número de serie del dispositivo.
    3. Genere un certificado para el dispositivo y regístrelo en AWS.
4. Cargue el certificado de entidad de certificación y el certificado de dispositivo en el dispositivo siguiendo las instrucciones de Aprovisionamiento de claves en modo desarrollador.

Monitorización de mensajes de MQTT en la nube de AWS

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT.

3. En Tema de suscripción, introduzca *your-thing-name*/example/topic y, a continuación, elija Suscribirse al tema.

Creación, instalación y ejecución del proyecto de demostración de FreeRTOS con el script idf.py

Puede utilizar la utilidad IDF de Espressif (`idf.py`) para generar los archivos de creación, crear el binario de la aplicación e instalar la placa.

> **Note**
>
> Algunas configuraciones pueden requerir que utilice la opción de puerto "`-p port-name`" con `idf.py` para especificar el puerto correcto, como en el siguiente ejemplo.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Creación e instalación de FreeRTOS en Windows, Linux y macOS (ESP-IDF v4.2)

1. Desplácese hasta la raíz del directorio de descargas de FreeRTOS.
2. En una ventana de línea de comandos, introduzca el siguiente comando para añadir las herramientas ESP-IDF a la ruta del terminal:

   Windows (aplicación "Comando")

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (aplicación "ESP-IDF 4.x CMD")

   (Esto ya se hizo cuando abrió aplicación).

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Configure cmake en el directorio `build` y cree la imagen del firmware con el siguiente comando.

```
idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
 build
```

Debería ver un resultado como el del siguiente ejemplo.

```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
   Warn about uninitialized values.
   -- Found Git: /usr/bin/git (found version "2.17.0")
   -- Building empty aws_iot component due to configuration
   -- Component names: ...
   -- Component paths: ...

   ... (more lines of build system output)

   [527/527] Generating hello-world.bin
   esptool.py v2.3.1

   Project build complete. To flash, run this command:
   ../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600
 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
hello-world.bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/
partition_table/partition-table.bin
   or run 'idf.py -p PORT flash'
```

Si no hay errores, la creación generará los archivos .bin binarios del firmware.

4. Borre la memoria flash de la placa de desarrollo con el siguiente comando.

```
idf.py erase_flash
```

5. Utilice el script `idf.py` para instalar el archivo binario de la aplicación en la placa.

```
idf.py flash
```

6. Supervise la salida del puerto serie de la placa con el siguiente comando.

```
idf.py monitor
```

> **Note**
>
> - Puede combinar estos comandos como se muestra en el siguiente ejemplo.

```
idf.py erase_flash flash monitor
```

- Para determinadas configuraciones de máquinas host, debe especificar el puerto en el que se va a instalar la placa, como en el siguiente ejemplo.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

Creación e instalación de FreeRTOS con CMake

Además de usar el script `idf.py` proporcionado por el SDK de IDF para crear y ejecutar su código, también puede crear el proyecto con CMake. Actualmente es compatible con Unix Makefile y el sistema de creación Ninja.

Para crear e instalar el proyecto

1. En una ventana de línea de comandos, navegue hasta el directorio de descargas de FreeRTOS.
2. Ejecute el siguiente script para añadir las herramientas ESP-IDF a la ruta de su intérprete de comandos.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. Utilice el siguiente comando para generar los archivos de creación.

   Con Unix Makefiles

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-
   esp32 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -
   DAFR_ENABLE_TESTS=0
   ```

Con Ninja

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-
esp32 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -
DAFR_ENABLE_TESTS=0 -GNinja
```

4. Borre la instalación y, a continuación, instale la placa.

   Con Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Con Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Información adicional

Para obtener más información sobre cómo utilizar y solucionar problemas de las placas Espressif ESP32, consulte los siguientes temas:

- Uso de FreeRTOS en su propio proyecto de CMake para ESP32
- Solución de problemas
- Debugging

Introducción a Espressif ESP32-S2

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un

proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

> **Note**
>
> Para explorar cómo integrar las bibliotecas modulares y demostraciones de FreeRTOS en su propio proyecto Espressif IDF, consulte nuestra integración de referencia destacada para la plataforma ESP32-C3.

Este tutorial le muestra cómo empezar a utilizar las placas de desarrollo Espressif ESP32-S2 SoC y ESP32-S2-Saola-1.

Información general

Este tutorial le guiará a través de los siguientes pasos:

1. Conecte la placa a un equipo host.
2. Instale el software en su equipo host para desarrollar y depurar las aplicaciones integradas de la placa del microcontrolador.
3. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria
4. Cargue la imagen binaria de la aplicación en su placa y, a continuación, ejecute la aplicación.
5. Monitorice y depure la aplicación en ejecución mediante una conexión serie.

Requisitos previos

Antes de comenzar con FreeRTOS en la placa de Espressif, tiene que configurar la cuenta de AWS y los permisos.

Registro para obtener una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Cómo registrarse en 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.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para realizar tareas que requieran acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación luego de completar 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.

Crear un usuario administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

Protección de su Usuario raíz de la cuenta de AWS

1. Inicie sesión en la AWS Management Console como propietario de cuenta, elija Usuario raíz e ingrese el email de su Cuenta de AWS. 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 raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario administrativo

1. Activar IAM Identity Center

   Para conocer las instrucciones, consulte Habilitar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center.
2. En IAM Identity Center, otorga acceso administrativo a un usuario administrativo.

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

Cómo iniciar sesión como usuario administrativo

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

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

Para dar acceso, añada permisos a los usuarios, grupos o roles:

- Usuarios y grupos en AWS IAM Identity Center:

  Cree un conjunto de permisos. Siga las instrucciones de Creación de un conjunto de permisos en la Guía del usuario de AWS IAM Identity Center.
- Usuarios administrados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones de Creación de un rol para un proveedor de identidades de terceros (federación) en la Guía del usuario de IAM.
- Usuarios de IAM:
  - Cree un rol que el usuario pueda aceptar. Siga las instrucciones descritas en Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.
  - (No recomendado) Asocie una política directamente a un usuario o añada un usuario a un grupo de usuarios. Siga las instrucciones descritas en Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Introducción

> **Note**
>
> Los comandos de Linux de este tutorial requieren que utilice el intérprete de comandos Bash.

1. Configure el hardware de Espressif.

   Para obtener información acerca de cómo configurar el hardware de la placa de desarrollo ESP32-S2, consulte ESP32-S2-Saola-1.

   > **Important**
   >
   > Cuando llegue a la sección Introducción de las guías de Espressif, deténgase y vuelva a las instrucciones en esta página.

2. Descargue Amazon FreeRTOS desde. GitHub (Para ver las instrucciones, consulte el archivo README.md).
3. Configure el entorno de desarrollo.

   Debe instalar una cadena de herramientas para comunicarse con la placa. Espressif proporciona el ESP-IDF para desarrollar software para sus placas. Dado que el ESP-IDF tiene su propia versión del kernel de FreeRTOS integrada como componente, Amazon FreeRTOS incluye una versión personalizada del ESP-IDF v4.2 en la que se ha eliminado el kernel de FreeRTOS. Esto soluciona los problemas relacionados con los archivos duplicados al compilar. Para usar la versión personalizada del ESP-IDF v4.2 incluida en Amazon FreeRTOS, siga las instrucciones que aparecen a continuación para el sistema operativo de su máquina host.

   Windows

   1. Descargue el instalador en línea universal de ESP-IDF para Windows.
   2. Ejecute el instalador en línea universal.
   3. Cuando llegue al paso Descargar o usar ESP-IDF, seleccione Usar un directorio ESP-IDF existente y establezca Elegir directorio ESP-IDF existente en *`freertos`*`/vendors/espressif/esp-idf`.
   4. Complete la instalación.

   macOS

   1. Siga las instrucciones que se indican en Requisitos previos para la configuración estándar de la cadena de herramientas (ESP-IDF v4.2) para macOS.

> **Important**
>
> Cuando llegue a las instrucciones de “Obtención de ESP-IDF” en Pasos siguientes, deténgase y vuelva a las instrucciones de esta página.

2. Abra una ventana de línea de comandos.
3. Navegue hasta el directorio de descargas de FreeRTOS y ejecute el siguiente script para descargar e instalar la cadena de herramientas espressif para su plataforma.

```
vendors/espressif/esp-idf/install.sh
```

4. Añada las herramientas de la cadena de herramientas ESP-IDF a la ruta de su terminal con el siguiente comando.

```
source vendors/espressif/esp-idf/export.sh
```

Linux

1. Siga las instrucciones que se indican en Requisitos previos para la configuración estándar de la cadena de herramientas (ESP-IDF v4.2) para Linux.

> **Important**
>
> Cuando llegue a las instrucciones de “Obtención de ESP-IDF” en Pasos siguientes, deténgase y vuelva a las instrucciones de esta página.

2. Abra una ventana de línea de comandos.
3. Navegue hasta el directorio de descargas de FreeRTOS y ejecute el siguiente script para descargar e instalar la cadena de herramientas Espressif para su plataforma.

```
vendors/espressif/esp-idf/install.sh
```

4. Añada las herramientas de la cadena de herramientas ESP-IDF a la ruta de su terminal con el siguiente comando.

```
source vendors/espressif/esp-idf/export.sh
```

4. Establezca una conexión serie.

   a. Para establecer una conexión en serie entre su máquina host y el DevKit ESP32-C, instale los controladores CP210x USB to UART Bridge VCP. Puede descargar estos controladores de Silicon Labs.

   b. Siga los pasos para establecer una conexión serie con ESP32.

   c. Después de establecer una conexión serie, anote el puerto serie de la conexión de la placa. Lo necesita para instalar la demostración.

Configuración de las aplicaciones de demostración de FreeRTOS

Para este tutorial, el archivo de configuración de FreeRTOS se encuentra en *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h. (Por ejemplo, si se elige AFR_BOARD espressif.esp32_devkitc, el archivo de configuración se encuentra en *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h).

1. Si ejecuta macOS o Linux, abra un símbolo del terminal. Si utiliza Windows, abra la aplicación “ESP-IDF 4.x CMD” (si incluyó esta opción al instalar la cadena de herramientas de ESP-IDF) o la aplicación de “símbolo del sistema” en caso contrario.
2. Para verificar que tiene instalado Python3, ejecute lo siguiente:

   ```
   python --version
   ```

   Se muestra la versión instalada. Si no tiene instalado Python 3.0.1 o una versión posterior, puede instalarlo desde el sitio web de Python.
3. Necesita la interfaz de la línea de comandos (CLI) de AWS para ejecutar los comandos de AWS IoT. Si ejecuta Windows, utilice el comando easy_install awscli para instalar la CLI de AWS en la aplicación de “Comando” o “ESP-IDF 4.x CMD”.

   Si ejecuta macOS o Linux, consulte Instalación de la CLI de AWS.
4. Ejecute

   ```
   aws configure
   ```

y configure la CLI de AWS con su ID de clave de acceso de AWS, la clave de acceso secreta y la región predeterminada de AWS. Para obtener más información, consulte Configuración de la CLI de AWS.

5. Utilice el comando siguiente para instalar el SDK de AWS para Python (boto3):

   - En Windows, en la aplicación de “Comando” o “ESP-IDF 4.x CMD”, ejecute

     ```
     easy_install boto3
     ```

   - En macOS o Linux, ejecute

     ```
     pip install tornado nose --user
     ```

     y, a continuación, ejecute

     ```
     pip install boto3 --user
     ```

FreeRTOS incluye el script `SetupAWS.py` para facilitar la configuración de su placa Espressif para conectar a AWS IoT.

Para ejecutar el script de configuración

1. Para configurar el script, abra *`freertos`*`/tools/aws_config_quick_start/configure.json` y defina los siguientes atributos:

   `afr_source_dir`

   Ruta completa del directorio *`freertos`* de su equipo. Asegúrese de que utiliza barras diagonales para especificar esta ruta.

   `thing_name`

   El nombre que desea asignar al objeto de AWS IoT que representa la placa.

   `wifi_ssid`

   El SSID de su red wifi.

`wifi_password`

La contraseña para su red wifi.

`wifi_security`

El tipo de seguridad para su red wifi. Los siguientes son tipos de seguridad válidos:

- `eWiFiSecurityOpen` (abierta, sin seguridad)
- `eWiFiSecurityWEP` (seguridad WEP)
- `eWiFiSecurityWPA` (seguridad WPA)
- `eWiFiSecurityWPA2` (seguridad WPA2)

2. Si ejecuta macOS o Linux, abra un símbolo del terminal. Si ejecuta Windows, abra la aplicación de “ESP-IDF 4.x CMD” o “Comando”.
3. Vaya al directorio *`freertos`*`/tools/aws_config_quick_start` y ejecute

```
python SetupAWS.py setup
```

El script hace lo siguiente:

- Crea un objeto de AWS IoT, un certificado y una política.
- Asocia la política de AWS IoT para el certificado y el certificado al objeto de AWS IoT.
- Rellena el archivo `aws_clientcredential.h` con su punto de conexión de AWS IoT, el SSID de la red wifi y las credenciales.
- Formatea el certificado y la clave privada y los escribe en el archivo de encabezado `aws_clientcredential_keys.h`.

> **Note**
>
> El certificado tiene codificación fija únicamente con fines ilustrativos. Las aplicaciones de producción deben almacenar estos archivos en un lugar seguro.

Para obtener más información sobre `SetupAWS.py`, consulte el archivo `README.md` en el directorio *`freertos`*`/tools/aws_config_quick_start`.

Monitorización de mensajes de MQTT en la nube de AWS

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT.
3. En Tema de suscripción, escriba *your-thing-name*`/example/topic` y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje "¡Hola, mundo!" enviado varias veces al tema al que se ha suscrito.

Creación, instalación y ejecución del proyecto de demostración de FreeRTOS con el script idf.py

Puede utilizar la utilidad IDF de Espressif para generar los archivos de creación, crear el binario de la aplicación e instalar la placa.

Creación e instalación de FreeRTOS en Windows, Linux y macOS (ESP-IDF v4.2)

Use el script `idf.py` para crear el proyecto e instalar los archivos binarios en su dispositivo.

> **Note**
>
> Algunas configuraciones pueden requerir que utilice la opción de puerto `-p port-name` con `idf.py` para especificar el puerto correcto, como en el siguiente ejemplo.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Para crear e instalar el proyecto

1. Desplácese hasta la raíz del directorio de descargas de FreeRTOS.
2. En una ventana de línea de comandos, introduzca el siguiente comando para añadir las herramientas ESP-IDF a la ruta del terminal:

Windows (aplicación “Comando”)

```
vendors\espressif\esp-idf\export.bat
```

Windows (aplicación “ESP-IDF 4.x CMD”)

(Esto ya se hizo cuando abrió aplicación).

Linux/macOS

```
source vendors/espressif/esp-idf/export.sh
```

3. Configure cmake en el directorio `build` y cree la imagen del firmware con el siguiente comando.

```
idf.py -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
```

Debería ver un resultado como el del siguiente ejemplo.

```
Executing action: all (aliases: build)
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1
 -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -
DCCACHE_ENABLE=0 /path/to/hello_world"...
   -- The C compiler identification is GNU 8.4.0
   -- The CXX compiler identification is GNU 8.4.0
   -- The ASM compiler identification is GNU

   ... (more lines of build system output)

   [1628/1628] Generating binary image from built executable
   esptool.py v3.0
   Generated /path/to/hello_world/build/aws_demos.bin

   Project build complete. To flash, run this command:
   esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset --chip
 esp32s2  write_flash --flash_mode dio --flash_size detect --flash_freq 80m 0x1000
 build/bootloader/bootloader.bin 0x8000 build/partition_table/partition-table.bin
 0x16000 build/ota_data_initial.bin 0x20000 build/aws_demos.bin
   or run 'idf.py -p (PORT) flash'
```

Si no hay errores, la creación generará los archivos .bin binarios del firmware.

4. Borre la memoria flash de la placa de desarrollo con el siguiente comando.

```
idf.py erase_flash
```

5. Utilice el script `idf.py` para instalar el archivo binario de la aplicación en la placa.

```
idf.py flash
```

6. Supervise la salida del puerto serie de la placa con el siguiente comando.

```
idf.py monitor
```

> **Note**
>
> - Puede combinar estos comandos como se muestra en el siguiente ejemplo.
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - Para determinadas configuraciones de máquinas host, debe especificar el puerto en el que se va a instalar la placa, como en el siguiente ejemplo.
>
>   ```
>   idf.py erase_flash flash monitor -p /dev/ttyUSB1
>   ```

Creación e instalación de FreeRTOS con CMake

Además de usar el script `idf.py` proporcionado por el SDK de IDF para crear y ejecutar su código, también puede crear el proyecto con CMake. Actualmente es compatible con Unix Makefile y el sistema de creación Ninja.

Para crear e instalar el proyecto

1. En una ventana de línea de comandos, navegue hasta el directorio de descargas de FreeRTOS.
2. Ejecute el siguiente script para añadir las herramientas ESP-IDF a la ruta de su intérprete de comandos.

   - Windows

```
vendors\espressif\esp-idf\export.bat
```

- Linux/macOS

```
source vendors/espressif/esp-idf/export.sh
```

3. Utilice el siguiente comando para generar los archivos de creación.

   - Con Unix Makefiles

```
cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
```

   - Con Ninja

```
cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
```

4. Compilar el proyecto.

   - Con Unix Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY -j8
```

   - Con Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. Borre la instalación y, a continuación, instale la placa.

   - Con Unix Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
make -C ./YOUR_BUILD_DIRECTORY flash
```

   - Con Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

Información adicional

Para obtener más información sobre cómo utilizar y solucionar problemas de las placas Espressif ESP32, consulte los siguientes temas:

- Uso de FreeRTOS en su propio proyecto de CMake para ESP32
- Solución de problemas
- Debugging

Introducción al kit de conectividad de IoT XMC4800 de Infineon

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción al kit de conectividad de IoT Infineon XMC4800. Si todavía no tiene el Kit de conectividad de IoT XMC4800, consulte el Catálogo de dispositivos de socios de AWS para comprar uno de nuestro socio.

Si desea abrir una conexión serie con la placa para ver información de registro y depuración, necesita un convertidor USB a serie de 3,3 V, además del kit de conectividad de IoT XMC4800. El CP2104 es un convertidor USB a serie común de amplia disponibilidad en placas como la CP2104 Friend de Adafruit.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
2. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
3. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.
4. Interacción con la aplicación que se ejecuta en la placa con una conexión serie para fines de monitorización y depuración.

Configure el entorno de desarrollo.

FreeRTOS utiliza el entorno de desarrollo DAVE de Infineon para programar el XMC4800. Antes de comenzar, tendrá que descargar e instalar DAVE y algunos controladores J-Link para comunicarse con el depurador incorporado.

Instale DAVE

1. Vaya a la página de descarga de software DAVE de Infineon.
2. Elija el paquete de DAVE para su sistema operativo y envíe su información de registro. Después de registrarse en Infineon, debería recibir un mail de confirmación con un enlace para descargar un archivo .zip.
3. Descargue el archivo .zip del paquete de Dave (`DAVE_version_os_date.zip`) y descomprímalo a la ubicación en la que desee instalar DAVE (por ejemplo, `C:\DAVE4`).

   > **Note**
   >
   > Algunos usuarios de Windows han informado de problemas al usar el explorador de Windows para descomprimir el archivo. Le recomendamos que utilice un programa de terceros como 7-Zip.

4. Para lanzar DAVE, ejecute el archivo ejecutable que podrá encontrar en la carpeta descomprimida `DAVE_version_os_date.zip`.

(Para obtener más información, consulte la DAVE Quick Start Guide).

Instale controladores Segger J-Link

Para comunicarse con la sonda de depuración incorporada de la placa XMC4800 Relax EtherCAT, necesita los controladores que se incluyen en el paquete de documentación y software de J-Link. Puede descargar el paquete de documentación y software de J-Link a través de la página de descarga de software de J-Link de Segger.

Establecimiento de una conexión serie

La configuración de una conexión serial es opcional, pero se recomienda. Una conexión serie permite a la placa enviar información de registro y depuración en un formato que puede ver en su equipo de desarrollo.

La aplicación de demostración XMC4800 utiliza una conexión serie UART en los pines P0.0 y P0.1, que se etiquetan en la serigrafía de la placa XMC4800 Relax EtherCAT. Para configurar una conexión serie:

1. Conecte el pin con la etiqueta “RX<P0.0” al pin “TX” de su convertidor USB a serie.
2. Conecte el pin con la etiqueta “TX>P0.1” al pin “RX” de su convertidor USB a serie.
3. Conecte el pin de conexión a tierra del convertidor serie a uno de los pines etiquetados “GND” en la placa. Los dispositivos deben compartir una conexión a tierra común.

La alimentación se suministra desde el puerto de depuración USB, así que no conecte el pin de tensión positiva del adaptador serie a la placa.

> **Note**
>
> Algunos cables serie utilizan un nivel de señalización de 5 V. El XMC4800 board and the Wi-Fi Haga clic en memoria requieren un 3,3 V. No utilice el puente IOREF de la placa para cambiar las señales de la placa a 5 V.

Con el cable conectado, puede abrir una conexión serie en un emulador de terminal como por ejemplo GNU Screen. La velocidad en baudios se establece en 115 200 de forma predeterminada con 8 bits de datos, sin paridad, y 1 bit de parada.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar la demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje “¡Hola, mundo!” enviado varias veces al tema al que se ha suscrito.

Creación y ejecución del proyecto de demostración de FreeRTOS

Importación de la demostración de FreeRTOS a DAVE

1. Inicie DAVE.
2. En DAVE, elija File (Archivo), Import (Importar). En la ventana Import (Importar), expanda la carpeta Infineon, elija DAVE Project (Proyecto DAVE) y, a continuación, elija Next (Siguiente).
3. En la ventana Import DAVE Projects (Importar proyectos de DAVE), elija Select Root Directory (Seleccionar directorio raíz), elija Browse (Examinar) y, a continuación, elija el proyecto de demostración XMC4800.

   En el directorio en el que ha descomprimido la descarga de FreeRTOS, el proyecto de demostración se encuentra en `projects/infineon/xmc4800_iotkit/dave4/aws_demos`.

   Asegúrese de que Copy Projects Into Workspace (Copiar proyectos en Workspace) no esté marcada).
4. Seleccione Finalizar.

   El proyecto `aws_demos` debe importarse a su espacio de trabajo y activarse.
5. En el menú Project (Proyecto) elija Build Active Project (Compilar proyecto activo).

   Asegúrese de que el proyecto se crea sin errores.

Ejecución del proyecto de demostración de FreeRTOS

1. Utilice un cable USB para conectar su kit de conectividad de IoT XMC4800 a su equipo. La placa tiene dos conectores microUSB. Utilice el denominado “X101”, donde Debug aparece al lado en la serigrafía de la placa.
2. En el menú Project (Proyecto), elija Rebuild Active Project (Reconstruir proyecto activo) para reconstruir `aws_demos` y garantizar que se recogen los cambios realizados a la configuración.
3. Desde Project Explorer (Explorador de proyectos), haga clic con el botón derecho del ratón en `aws_demos`, elija Debug As (Depurar como) y, a continuación, elija DAVE C/C++ Application (Aplicación DAVE C/C++).
4. Haga doble clic en GDB SEGGER J-Link Debugging (Depuración de J-Link de SEGGER con GDB) para crear una confirmación de depuración. Elija Debug (Depuración).
5. Cuando el depurador se detenga en el punto de ruptura en `main()`, desde el menú Run (Ejecutar), elija Resume (Reanudar).

En la consola de AWS IoT, el cliente de MQTT de los pasos 4-5 debe mostrar los mensajes de MQTT enviados por el dispositivo. Si utiliza la conexión serie, verá algo como esto en la salida UART:

```
0 0 [Tmr Svc] Starting key provisioning...
1 1 [Tmr Svc] Write root certificate...
2 4 [Tmr Svc] Write device private key...
3 82 [Tmr Svc] Write device certificate...
4 86 [Tmr Svc] Key provisioning done...
5 291 [Tmr Svc] Wi-Fi module initialized. Connecting to AP...
.6 8046 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
7 8058 [Tmr Svc] IP Address acquired [IP Address]
8 8058 [Tmr Svc] Creating MQTT Echo Task...
9 8059 [MQTTEcho] MQTT echo attempting to connect to [MQTT Broker].
...10 23010 [MQTTEcho] MQTT echo connected.
11 23010 [MQTTEcho] MQTT echo test echoing task created.
.12 26011 [MQTTEcho] MQTT Echo demo subscribed to iotdemo/#
13 29012 [MQTTEcho] Echo successfully published 'Hello World 0'
.14 32096 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
.15 37013 [MQTTEcho] Echo successfully published 'Hello World 1'
16 40080 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
.17 45014 [MQTTEcho] Echo successfully published 'Hello World 2'
.18 48091 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
.19 53015 [MQTTEcho] Echo successfully published 'Hello World 3'
```

```
.20 56087 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
.21 61016 [MQTTEcho] Echo successfully published 'Hello World 4'
22 64083 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
.23 69017 [MQTTEcho] Echo successfully published 'Hello World 5'
.24 72091 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
.25 77018 [MQTTEcho] Echo successfully published 'Hello World 6'
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
.27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
.28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
.29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
.30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
.31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
.33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
.34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
.35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
.37 122068 [MQTTEcho] MQTT echo demo finished.
38 122068 [MQTTEcho] ----Demo finished----
```

Creación de la demostración de FreeRTOS con CMake

Si prefiere no utilizar un IDE para el desarrollo de FreeRTOS, también puede usar CMake para crear y ejecutar las aplicaciones de demostración o las aplicaciones que ha desarrollado con herramientas de depuración y editores de código de terceros.

> **Note**
>
> En esta sección se explica el uso de CMake en Windows con MingW como el sistema de compilación nativo. Para obtener más información acerca de cómo utilizar CMake con otros sistemas operativos y opciones, consulte Uso de CMake con FreeRTOS. (MinGW es un entorno de desarrollo minimalista para aplicaciones nativas de Microsoft Windows).

Para crear la demostración de FreeRTOS con CMake

1. Configure la cadena de herramientas GNU Arm Embedded Toolchain.
   a. Descargue una versión de Windows de la cadena de herramientas de la página de descargas de Arm Embedded Toolchain.

> **Note**
>
> Le recomendamos que descargue una versión que no sea "8-2018-q4-major", debido a un error comunicado con la utilidad “objcopy” en esa versión.

b. Abra el instalador de la cadena de herramientas descargada y siga las instrucciones del asistente para instalar la cadena de herramientas.

> **Important**
>
> En la última página del asistente de instalación, seleccione Add path to environment variable (Añadir ruta a variable de entorno) para añadir la ruta de la cadena de herramientas a la variable de entorno de ruta del sistema.

2. Instale CMake y MingW.

   Para obtener instrucciones, consulte Requisitos previos de CMake.

3. Cree una carpeta para contener los archivos de compilación generados (*carpeta-compilación*).

4. Cambie los directorios por el directorio de descargas de FreeRTOS (*freertos*) y utilice el siguiente comando para generar los archivos de creación:

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. Cambie los directorios al directorio de compilación (*carpeta-compilación*) y utilice el siguiente comando para compilar el binario:

```
cmake --build . --parallel 8
```

   Este comando compila el binario de salida `aws_demos.hex` en el directorio de compilación.

6. Actualice y ejecute la imagen con JLINK.

   a. Desde el directorio de compilación (*carpeta-compilación*), utilice los siguientes comandos para crear un script de actualización:

```
echo loadfile aws_demos.hex > flash.jlink
```

```
echo r >> flash.jlink
```

```
echo g >> flash.jlink
```

```
echo q >> flash.jlink
```

b. Actualice la imagen mediante el ejecutable de JLNIK.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

Los registros de la aplicación deben ser visibles a través de la conexión serie que ha establecido con la placa.

Solución de problemas

Si aún no lo ha hecho, asegúrese de configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos.

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción al kit de conectividad de IoT XMC4800 y OPTIGA Trust X de Infineon

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción al kit de conectividad de IoT XMC4800 y el elemento seguro de OPTIGA Trust X de Infineon. En comparación con el tutorial Introducción al kit de conectividad de IoT XMC4800 de Infineon, esta guía le muestra cómo proporcionar credenciales seguras mediante un elemento seguro de OPTIGA Trust X de Infineon.

Necesitará el siguiente hardware:

1. MCU del host - Kit de conectividad de IoT XMC4800 de Infineon, consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestros socios.
2. Paquete de extensión de seguridad:
   - Elemento seguro: OPTIGA Trust X de Infineon.

     Consulte el Catálogo de dispositivos de socios de AWS para adquirirlos de nuestro socio.
   - Placa de personalización: placa de personalización OPTIGA de Infineon.
   - Placa del adaptador: adaptador MyIoT de Infineon.

Para seguir estos pasos, debe abrir una conexión serie con la placa para ver la información de registro y depuración. (Uno de los pasos requiere copiar una clave pública de la salida de depuración en serie de la placa y pegarla en un archivo). Para ello, necesita un convertidor USB/serie de 3,3 V además del kit de conectividad de IoT XMC4800. El convertidor USB a serie JBtek EL-PN-47310126 es compatible con esta demostración. También necesitará tres cables puente macho a macho (para la recepción [RX], la transmisión [TX] y la conexión a tierra [GND]) para conectar el cable de serie a la placa del adaptador MyIoT de Infineon.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Opción n.º 2: generación de claves privadas integrada. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *`freertos`*.

Información general

Este tutorial contiene los siguientes pasos:

1. Instale el software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa del microcontrolador.
2. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria
3. Cargue la imagen binaria de la aplicación en su placa y, a continuación, ejecute la aplicación.
4. Interactúe con la aplicación que se está ejecutando en la placa con una conexión serie para fines de monitorización y depuración.

Configure el entorno de desarrollo.

FreeRTOS utiliza el entorno de desarrollo DAVE de Infineon para programar el XMC4800. Antes de comenzar, descargue e instale DAVE y algunos controladores J-Link para comunicarse con el depurador incorporado.

Instale DAVE

1. Vaya a la página de descarga de software DAVE de Infineon.
2. Elija el paquete de DAVE para su sistema operativo y envíe su información de registro. Después de registrarse, debe recibir un email de confirmación con un enlace para descargar un archivo .zip.
3. Descargue el archivo .zip del paquete de Dave (`DAVE_version_os_date.zip`) y descomprímalo a la ubicación en la que desee instalar DAVE (por ejemplo, `C:\DAVE4`).

   > **Note**
   >
   > Algunos usuarios de Windows han informado de problemas al usar el explorador de Windows para descomprimir el archivo. Le recomendamos que utilice un programa de terceros como 7-Zip.

4. Para lanzar DAVE, ejecute el archivo ejecutable que podrá encontrar en la carpeta descomprimida `DAVE_version_os_date.zip`.

(Para obtener más información, consulte la DAVE Quick Start Guide).

Instale controladores Segger J-Link

Para comunicarse con la sonda de depuración incorporada del kit de conectividad de IoT XMC4800, necesita los controladores que se incluyen en el paquete de documentación y software de J-Link. Puede descargar el paquete de documentación y software de J-Link a través de la página de descarga de software de J-Link de Segger.

Establecimiento de una conexión serie

Conecte el cable convertidor USB a serie al adaptador Shield2Go de Infineon. Esto permite que la placa envíe información de registro y depuración en un formato que puede ver en su equipo de desarrollo. Para configurar una conexión serie:

1. Conecte el pin RX al pin TX del convertidor USB a serie.

2. Conecte el pin TX al pin RX del convertidor USB a serie.
3. Conecte el pin de conexión a tierra del convertidor de serie a uno de los pines GND de la placa. Los dispositivos deben compartir una conexión a tierra común.

La alimentación se suministra desde el puerto de depuración USB, así que no conecte el pin de tensión positiva del adaptador serie a la placa.

> **Note**
>
> Algunos cables serie utilizan un nivel de señalización de 5 V. El XMC4800 board and the Wi-Fi Haga clic en memoria requieren un 3,3 V. No utilice el puente IOREF de la placa para cambiar las señales de la placa a 5 V.

Con el cable conectado, puede abrir una conexión serie en un emulador de terminal como por ejemplo GNU Screen. La velocidad en baudios se establece en 115 200 de forma predeterminada con 8 bits de datos, sin paridad, y 1 bit de parada.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje “¡Hola, mundo!” enviado varias veces al tema al que se ha suscrito.

Creación y ejecución del proyecto de demostración de FreeRTOS

Importación de la demostración de FreeRTOS a DAVE

1. Inicie DAVE.

2. En DAVE, elija File (Archivo) y, a continuación, elija Import (Importar). Expanda la carpeta Infineon, elija DAVE Project (Proyecto DAVE) y, a continuación, elija Next (Siguiente).
3. En la ventana Import DAVE Projects (Importar proyectos de DAVE), elija Select Root Directory (Seleccionar directorio raíz), elija Browse (Examinar) y, a continuación, elija el proyecto de demostración XMC4800.

   En el directorio en el que ha descomprimido la descarga de FreeRTOS, el proyecto de demostración se encuentra en `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`.

   Asegúrese de que Copy Projects Into Workspace (Copiar proyectos en Workspace) no esté marcado.
4. Elija Finalizar.

   El proyecto `aws_demos` debe importarse a su espacio de trabajo y activarse.
5. En el menú Project (Proyecto) elija Build Active Project (Compilar proyecto activo).

   Asegúrese de que el proyecto se crea sin errores.

Ejecución del proyecto de demostración de FreeRTOS

1. En el menú Project (Proyecto), elija Rebuild Active Project (Reconstruir proyecto activo) para reconstruir `aws_demos` y confirmar que se recogen los cambios realizados a la configuración.
2. Desde Project Explorer (Explorador de proyectos), haga clic con el botón derecho del ratón en `aws_demos`, elija Debug As (Depurar como) y, a continuación, elija DAVE C/C++ Application (Aplicación DAVE C/C++).
3. Haga doble clic en GDB SEGGER J-Link Debugging (Depuración de J-Link de SEGGER con GDB) para crear una confirmación de depuración. Elija Debug (Depuración).
4. Cuando el depurador se detenga en el punto de ruptura en `main()`, desde el menú Run (Ejecutar), elija Resume (Reanudar).

En este punto, continúe con el paso de extracción de clave pública que figura en Opción n.º 2: generación de claves privadas integrada. Una vez completados todos los pasos, vaya a la consola de AWS IoT. El cliente de MQTT que ha configurado previamente debe mostrar los mensajes de MQTT enviados por el dispositivo. A través de la conexión serie del dispositivo, debe ver algo similar a esto en la salida UART:

```
0 0 [Tmr Svc] Starting key provisioning...
1 1 [Tmr Svc] Write root certificate...
2 4 [Tmr Svc] Write device private key...
3 82 [Tmr Svc] Write device certificate...
4 86 [Tmr Svc] Key provisioning done...
5 291 [Tmr Svc] Wi-Fi module initialized. Connecting to AP...
.6 8046 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
7 8058 [Tmr Svc] IP Address acquired [IP Address]
8 8058 [Tmr Svc] Creating MQTT Echo Task...
9 8059 [MQTTEcho] MQTT echo attempting to connect to [MQTT Broker].
...10 23010 [MQTTEcho] MQTT echo connected.
11 23010 [MQTTEcho] MQTT echo test echoing task created.
.12 26011 [MQTTEcho] MQTT Echo demo subscribed to iotdemo/#
13 29012 [MQTTEcho] Echo successfully published 'Hello World 0'
.14 32096 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
.15 37013 [MQTTEcho] Echo successfully published 'Hello World 1'
16 40080 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
.17 45014 [MQTTEcho] Echo successfully published 'Hello World 2'
.18 48091 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
.19 53015 [MQTTEcho] Echo successfully published 'Hello World 3'
.20 56087 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
.21 61016 [MQTTEcho] Echo successfully published 'Hello World 4'
22 64083 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
.23 69017 [MQTTEcho] Echo successfully published 'Hello World 5'
.24 72091 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
.25 77018 [MQTTEcho] Echo successfully published 'Hello World 6'
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
.27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
.28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
.29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
.30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
.31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
.33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
.34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
.35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
.37 122068 [MQTTEcho] MQTT echo demo finished.
38 122068 [MQTTEcho] ----Demo finished----
```

Creación de la demostración de FreeRTOS con CMake

En esta sección se explica el uso de CMake en Windows con MingW como el sistema de compilación nativo. Para obtener más información acerca de cómo utilizar CMake con otros sistemas operativos y opciones, consulte Uso de CMake con FreeRTOS. (MinGW es un entorno de desarrollo minimalista para aplicaciones nativas de Microsoft Windows).

Si prefiere no utilizar un IDE para el desarrollo de FreeRTOS, puede usar CMake para crear y ejecutar las aplicaciones de demostración o las aplicaciones que ha desarrollado con herramientas de depuración y editores de código de terceros.

Para crear la demostración de FreeRTOS con CMake

1. Configure la cadena de herramientas GNU Arm Embedded Toolchain.

   a. Descargue una versión de Windows de la cadena de herramientas de la página de descargas de Arm Embedded Toolchain.

      > **Note**
      >
      > Debido a un error notificado en la utilidad objcopy, recomendamos que descargue una versión distinta de "8-2018-q4-major".

   b. Abra el instalador de la cadena de herramientas descargado y siga las instrucciones del asistente.

   c. En la última página del asistente de instalación, seleccione Add path to environment variable (Añadir ruta a variable de entorno) para añadir la ruta de la cadena de herramientas a la variable de entorno de ruta del sistema.

2. Instale CMake y MingW.

   Para obtener instrucciones, consulte Requisitos previos de CMake.

3. Cree una carpeta para contener los archivos de compilación generados (*carpeta-compilación*).

4. Cambie los directorios por el directorio de descargas de FreeRTOS (*freertos*) y utilice el siguiente comando para generar los archivos de creación:

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S .
 -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. Cambie los directorios al directorio de compilación (*carpeta-compilación*) y utilice el siguiente comando para compilar el binario:

   ```
   cmake --build . --parallel 8
   ```

   Este comando compila el binario de salida `aws_demos.hex` en el directorio de compilación.

6. Actualice y ejecute la imagen con JLINK.

   a. Desde el directorio de compilación (*carpeta-compilación*), utilice los siguientes comandos para crear un script de actualización:

      ```
      echo loadfile aws_demos.hex > flash.jlink
      echo r >> flash.jlink
      echo g >> flash.jlink
      echo q >> flash.jlink
      ```

   b. Actualice la imagen mediante el ejecutable de JLNIK.

      ```
      JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
       flash.jlink
      ```

      Los registros de la aplicación deben ser visibles a través de la conexión serie que ha establecido con la placa. Continúe con el paso de extracción de clave pública en Opción n.º 2: generación de claves privadas integrada. Después de completar todos los pasos, vaya a la consola de AWS IoT. El cliente de MQTT que ha configurado previamente debe mostrar los mensajes de MQTT enviados por el dispositivo.

Solución de problemas

Para obtener información sobre la resolución de problemas, consulte Introducción a solución de problemas.

Introducción al Kit de inicio de AWS IoT MW32x

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un

proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

El Kit de inicio de AWS IoT es un kit de desarrollo basado en el 88MW320/88MW322, el último microcontrolador Cortex M4 integrado de NXP, que integra Wi-Fi 802.11b/g/n en un solo chip microcontrolador. El kit de desarrollo cuenta con la certificación de la FCC. Para obtener más información, consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestros socios. Los módulos 88MW320/88MW322 también cuentan con certificación FCC y están disponibles para personalización y venta de gran volumen.

Esta guía de introducción le muestra cómo compilar de forma cruzada su aplicación con el SDK en un equipo host y, a continuación, cargar el archivo binario generado en la placa con las herramientas incluidas en el SDK. Cuando la aplicación comience a ejecutarse en la placa, podrá depurarla o interactuar con ella desde la consola serie de su equipo host.

Ubuntu 16.04 es la plataforma host compatible para el desarrollo y la depuración. Es posible que pueda usar otras plataformas, pero no son compatibles oficialmente. Debe tener permisos para instalar el software en la plataforma host. Se requieren las siguientes herramientas externas para crear el SDK:

- Plataforma host Ubuntu 16.04
- Cadena de herramientas ARM, versión 4_9_2015q3
- IDE de Eclipse 4.9.0

La cadena de herramientas ARM es necesaria para realizar una compilación cruzada de la aplicación y el SDK. El SDK aprovecha las versiones más recientes de la cadena de herramientas para optimizar el tamaño de la imagen e incluir más funciones en menos espacio. En esta guía se asume que está utilizando la versión 4_9_2015q3 de la cadena de herramientas. No se recomienda utilizar versiones anteriores de la cadena de herramientas. El kit de desarrollo viene preinstalado con el firmware del proyecto de demostración Wireless Microcontroller.

Temas

- Configurar su hardware
- Configuración del entorno de desarrollo
- Creación y ejecución del proyecto de demostración de FreeRTOS

- Debugging
- Solución de problemas

Configurar su hardware

Conecte la placa MW32x al portátil mediante un cable mini-USB a USB. Conecte el cable mini-USB al único conector mini-USB presente en la placa. No es necesario un cambio de puente.

Si la placa está conectada a un ordenador portátil o de sobremesa, no necesita una fuente de alimentación externa.

Esta conexión USB proporciona lo siguiente:

- Consola de acceso a la placa. Se ha registrado un puerto tty/com virtual en el host de desarrollo que se puede utilizar para acceder a la consola.
- Acceso JTAG a la placa. Se puede utilizar para cargar o descargar imágenes de firmware en la RAM o la memoria flash de la placa, o con fines de depuración.

Configuración del entorno de desarrollo

Para fines de desarrollo, el requisito mínimo es la cadena de herramientas ARM y las herramientas incluidas en el SDK. En las siguientes secciones presentamos más detalles sobre la configuración de la cadena de herramientas de ARM.

Cadena de herramientas GNU

El SDK es compatible oficialmente con la cadena de herramientas del compilador GCC. La cadena de herramientas de compilación cruzada para GNU ARM está disponible en GNU Arm Embedded Toolchain 4.9-2015-q3-update.

El sistema de creación está configurado para usar la cadena de herramientas de GNU de forma predeterminada. Los archivos Make asumen que los binarios de la cadena de herramientas del compilador GNU están disponibles en la ruta del usuario y se pueden invocar desde los archivos Make. Los archivos Make también asumen que los nombres de archivo de los binarios de la cadena de herramientas de GNU llevan el prefijo `arm-none-eabi-`.

La cadena de herramientas de GCC se puede usar con GDB para depurar con OpenOCD (incluido con el SDK). Esto proporciona el software que interactúa con JTAG.

Recomendamos la versión 4_9_2015q3 de la cadena de herramientas gcc-arm-embedded.

Procedimiento de configuración de la cadena de herramientas en Linux

Siga estos pasos para configurar la cadena de herramientas de GCC en Linux.

1. Descargue el archivo tarball de la cadena de herramientas disponible en GNU Arm Embedded Toolchain 4.9-2015-q3-update. El archivo es `gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`.
2. Copie el archivo al directorio que elija. Asegúrese de que no hay espacios en el nombre del directorio.
3. Utilice el siguiente comando para extraer el archivo.

   ```
   tar -vxf filename
   ```

4. Añada la ruta de la cadena de herramientas instalada a la ruta del sistema. Por ejemplo, añada la siguiente línea al final del archivo `.profile` ubicado en el directorio `/home/user-name`.

   ```
   PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
   ```

> **Note**
>
> Las distribuciones de Ubuntu más recientes pueden incluir una versión Debian del compilador cruzado GCC. Si es así, debe eliminar el compilador cruzado nativo y seguir el procedimiento de configuración anterior.

Trabajo con un host de desarrollo de Linux

Se puede usar cualquier distribución moderna de escritorio de Linux, como Ubuntu o Fedora. Sin embargo, le recomendamos que actualice a la versión más reciente. Se ha comprobado que los siguientes pasos funcionan en Ubuntu 16.04 y se supone que está utilizando esa versión.

Instalación de paquetes

El SDK incluye un script que permite configurar rápidamente su entorno de desarrollo en una máquina Linux recién configurada. El script intenta detectar automáticamente el tipo de máquina e instalar el software adecuado, incluidas las bibliotecas C, la biblioteca USB, la biblioteca FTDI,

ncurses, python y latex. En esta sección, el nombre genérico del directorio *amzsdk_bundle-x.y.z* indica el directorio raíz del SDK de AWS. El nombre real del directorio puede ser diferente. Debe tener privilegios de raíz.

- Vaya al directorio *amzsdk_bundle-x.y.z/* y ejecute este comando.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

Evitar sudo

En esta guía, la operación `flashprog` utiliza el script `flashprog.py` para instalar la NAND de la placa, como se explica a continuación. Del mismo modo, la operación `ramload` utiliza el script `ramload.py` para copiar la imagen del firmware del host directamente a la RAM del microcontrolador, sin instalar la NAND.

Puede configurar su host de desarrollo de Linux para que realice las operaciones `flashprog` y `ramload` sin necesidad de utilizar el comando `sudo` cada vez. Para ello, ejecute el siguiente comando.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

> **Note**
>
> Debe configurar los permisos del host de desarrollo de Linux de esta manera para garantizar una experiencia de IDE de Eclipse fluida.

Configuración de la consola serie

Inserte el cable USB en la ranura USB del host Linux. Esto activa la detección del dispositivo. Debería ver mensajes como los siguientes en el archivo `/var/log/messages` o después de ejecutar el comando `dmesg`.

```
Jan 6 20:00:51 localhost kernel: usb 4-2: new full speed USB device using uhci_hcd and
 address 127
Jan 6 20:00:51 localhost kernel: usb 4-2: configuration #1 chosen from 1 choice
Jan 6 20:00:51 localhost kernel: ftdi_sio 4-2:1.0: FTDI USB Serial Device converter
 detected
```

```
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached
 to ttyUSB0
Jan 6 20:00:51 localhost kernel: ftdi_sio 4-2:1.1: FTDI USB Serial Device converter
 detected
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached
 to ttyUSB1
```

Compruebe que se hayan creado dos dispositivos ttyUSB. El segundo ttyUSB es la consola serie. En el ejemplo anterior, se denomina “ttyUSB1”.

En esta guía, utilizamos minicom para ver la salida de la consola en serie. También puede utilizar otros programas en serie, como `putty`. Ejecute el siguiente comando para ejecutar minicom en el modo de configuración.

```
minicom -s
```

En minicom, vaya a Configuración de puerto serie y capture los siguientes ajustes.

```
| A - Serial Device : /dev/ttyUSB1
| B - Lockfile Location : /var/lock
| C - Callin Program :
| D - Callout Program :
| E - Bps/Par/Bits : 115200 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control : No
```

Puede guardar estos ajustes en minicom para su uso posterior. La ventana minicom muestra ahora mensajes de la consola serie.

Seleccione la ventana de la consola serie y pulse la tecla Intro. Aparecerá un hash (#) en la pantalla.

> **Note**
>
> Las placas de desarrollo incluyen un dispositivo de silicio FTDI. El dispositivo FTDI expone dos interfaces USB para el host. La primera interfaz está asociada a la funcionalidad JTAG de la MCU y la segunda interfaz está asociada al puerto UARTx físico de la MCU.

Instalación de OpenOCD

OpenOCD es un software que proporciona depuración, programación en el sistema y pruebas de escaneo de límites para dispositivos de destino integrados.

Es necesaria la versión 0.9 de OpenOCD. También es necesario para la funcionalidad de Eclipse. Si se instaló una versión anterior, como la 0.7, en su host Linux, elimine ese repositorio con el comando adecuado para la distribución de Linux que esté utilizando.

Ejecute el comando estándar de Linux para instalar OpenOCD,

```
apt-get install openocd
```

Si el comando anterior no instala la versión 0.9 o posterior, utilice el siguiente procedimiento para descargar y compilar el código fuente de openocd.

Instalación de OpenOCD

1. Ejecute el siguiente comando para instalar libusb-1.0.

   ```
   sudo apt-get install libusb-1.0
   ```

2. Descargue el código fuente de la versión 0.9.0 de OpenOCD desde http://openocd.org/.
3. Extraiga openocd y acceda al directorio en el que lo extrajo.
4. Configure openocd con comando siguiente.

   ```
   ./configure --enable-ftdi --enable-jlink
   ```

5. Ejecute la utilidad make para compilar opencd.

   ```
   make install
   ```

Configuración de Eclipse

> **Note**
>
> En esta sección se supone que ha completado los pasos descritos en Evitar sudo.

Eclipse es el IDE preferido para el desarrollo y la depuración de aplicaciones. Proporciona un IDE completo y fácil de usar con soporte de depuración integrado, que incluye la depuración con reconocimiento de subprocesos. En esta sección se describe la configuración común de Eclipse para todos los hosts de desarrollo compatibles.

Para configurar Eclipse, realice el siguiente procedimiento:

1. Descargue e instale el entorno de ejecución de Java (JRE).

   Eclipse requiere que instale el JRE. Se recomienda instalarlo primero, aunque se puede instalar después de instalar Eclipse. La versión de JRE (32 bits o 64 bits) debe coincidir con la versión de Eclipse (32 bits o 64 bits) que instale. Puede descargar el JRE desde las descargas de Java SE Runtime Environment 8 en el sitio web de Oracle.
2. Descargue e instale el “IDE de Eclipse para desarrolladores de C/C++” desde http://www.eclipse.org. Se admiten las versiones de Eclipse 4.9.0 y superiores. La instalación solo requiere que extraiga el archivo descargado. Para iniciar la aplicación, debe ejecutar el ejecutable de Eclipse específico de la plataforma.

Creación y ejecución del proyecto de demostración de FreeRTOS

Hay dos formas de ejecutar el proyecto de demostración de FreeRTOS:

- Usar la línea de comandos.
- Usar el IDE de Eclipse.

En este tema se tratan ambas opciones.

Aprovisionar

- En función de si utiliza la aplicación de prueba o de demostración, configure los datos de aprovisionamiento en uno de los siguientes archivos:

  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  Por ejemplo:

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
```

```
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

> **Note**
>
> Puede introducir los siguientes valores de seguridad de Wi-Fi:
>
> - `eWiFiSecurityOpen`
> - `eWiFiSecurityWEP`
> - `eWiFiSecurityWPA`
> - `eWiFiSecurityWPA2`
>
> El SSID y la contraseña deberían estar entre comillas dobles.

Creación y ejecución de la demostración de FreeRTOS con la línea de comandos

1. Utilice el comando siguiente para empezar a crear la aplicación de demostración.

   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Asegúrese de obtener la misma salida que la que se muestra en el siguiente ejemplo.

```
marvell@pe-lt586:amzsdk$
marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -
Bbuild -DAFR_ENABLE_TESTS=0

====================Configuration for Amazon FreeRTOS====================
  Version:                 v1.2.4
  Git version:             AMZSDK_V1.2.r6.p1-12-gdd17d10

Target microcontroller:
  vendor:                  Marvell
  board:                   mw300_rd
  description:             Marvell Board for AmazonFreeRTOS
  family:                  Wireless Microcontroller
  data ram size:           512KB
  program memory size:     2MB

Host platform:
  OS:                      Linux-4.15.0-47-generic
  Toolchain:               arm-gcc
  Toolchain path:          /home/marvell/Software/gcc-arm-none-eabi-4_9-2015q3
  CMake generator:         Unix Makefiles

Amazon FreeRTOS modules:
  Modules to build:        kernel, freertos_plus_tcp, bufferpool, crypto, greengrass, mqtt
, ota, pkcs11, secure_sockets, shadow, tls, wifi
  Disabled by user:
  Disabled by dependency:  posix

  Available demos:         demo_key_provisioning, demo_logging, demo_mqtt_hello_world, dem
o_mqtt_pubsub, demo_tcp, demo_shadow, demo_greengrass, demo_ota
  Available tests:         test_crypto, test_greengrass, test_mqtt, test_ota, test_pkcs11,
 test_secure_sockets, test_tls, test_shadow, test_wifi, test_memory
=========================================================================

-- Configuring done
-- Generating done
-- Build files have been written to: /home/marvell/gerrit/amzsdk/build
marvell@pe-lt586:amzsdk$
```

2. Vaya al directorio de creación.

```
cd build
```

3. Ejecute la utilidad make para crear la aplicación.

```
make all -j4
```

Asegúrese de obtener la misma salida que la que se muestra en la siguiente figura:

4. Utilice los siguientes comandos para crear una aplicación de prueba.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=1
cd build
make all -j4
```

   Ejecute el comando `cmake` cada vez que cambie entre el `aws_demos project` y el `aws_tests project`.

5. Escriba la imagen del firmware en la memoria flash de la placa de desarrollo. El firmware se ejecutará después de restablecer la placa de desarrollo. Debe crear el SDK antes de instalar la imagen en el microcontrolador.

   a. Antes de instalar la imagen del firmware, prepare la memoria flash de la placa de desarrollo con los componentes comunes Layout y Boot2. Use los siguientes comandos.

   ```
   cd amzsdk_bundle-x.y.z
   ./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/
   marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/
   marvell/WMSDK/mw320/boot2/bin/boot2.bin
   ```

   El comando `flashprog` inicia lo siguiente:

- Diseño - Primero se indica a la utilidad flashprog que escriba un diseño en la memoria flash. El diseño es similar al de la información de partición de la memoria flash. El diseño predeterminado se encuentra en `/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.
- Boot2 - Este es el gestor de arranque utilizado por el WMSDK. El comando `flashprog` también escribe un cargador de arranque en la memoria flash. El cargador de arranque carga la imagen del firmware del microcontrolador después de que se haya instalado. Asegúrese de obtener la misma salida que la que se muestra en la siguiente figura.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245458s (115.728 KiB/s)
verified 29088 bytes in 0.350004s (81.160 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

b. El firmware utiliza el chipset Wi-Fi para su funcionalidad, y el chipset Wi-Fi tiene su propio firmware que también debe estar presente en la memoria flash. La utilidad `flashprog.py` se utiliza para actualizar el firmware de Wi-Fi de la misma forma que se utilizó para actualizar el gestor de arranque Boot2 y el firmware de la MCU. Utilice los siguientes comandos para instalar el firmware de Wi-Fi.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./
vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

Asegúrese de que el resultado del comando sea similar al de la siguiente figura.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary).....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

c. Utilice los siguientes comandos para instalar el firmware de la MCU.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. Restablezca la placa. Debería ver los registros de la aplicación de demostración.

e. Para ejecutar la aplicación de prueba, actualice el binario `aws_tests.bin` ubicado en el mismo directorio.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

La salida del comando debería ser similar a la que se muestra en la figura siguiente.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "mcufw" @0x6a000 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
Resetting board...
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
 select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
shutdown command invoked
Resetting board done...
```

6. Después de instalar el firmware y restablecer la placa, la aplicación de demostración debería iniciarse como se muestra en la siguiente figura.

```
Network connection successful.

Wi-Fi Connected to AP. Creating tasks which use network...
2 6293 [Startup Hook] Write certificate...
3 6296 [Startup Hook] Write device private key...
4 6362 [Startup Hook] Creating MQTT Echo Task...
6 11668 [MQTTEcho] MQTT echo connected.to connect to a2wtm15blvjjs8-ats.iot.us-east-2.amazonaws.com
7 11668 [MQTTEcho] MQTT echo test echoing task created.
8 11961 [MQTTEcho] MQTT Echo demo subscribed to freertos/demos/echo
9 12248 [MQTTEcho] Echo successfully published 'Hello World 0'
10 12591 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
11 17633 [MQTTEcho] Echo successfully published 'Hello World 1'
12 17927 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
13 22953 [MQTTEcho] Echo successfully published 'Hello World 2'
14 23276 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
15 28245 [MQTTEcho] Echo successfully published 'Hello World 3'
16 28575 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
17 33542 [MQTTEcho] Echo successfully published 'Hello World 4'
18 33980 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
19 38823 [MQTTEcho] Echo successfully published 'Hello World 5'
20 39279 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
21 44139 [MQTTEcho] Echo successfully published 'Hello World 6'
22 44501 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
23 49516 [MQTTEcho] Echo successfully published 'Hello World 7'
24 50270 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
25 54796 [MQTTEcho] Echo successfully published 'Hello World 8'
26 55129 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
27 60080 [MQTTEcho] Echo successfully published 'Hello World 9'
28 60389 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
29 65378 [MQTTEcho] Echo successfully published 'Hello World 10'
30 65998 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
31 70658 [MQTTEcho] Echo successfully published 'Hello World 11'
32 70964 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
33 75958 [MQTTEcho] MQTT echo demo finished.
34 75958 [MQTTEcho] ----Demo finished----
```

7. (Opcional) Como método alternativo para probar la imagen, utilice la utilidad flashprog para copiar la imagen del microcontrolador desde el host directamente a la RAM del microcontrolador. La imagen no se copia en la memoria flash, por lo que se perderá al reiniciar el microcontrolador.

   Cargar la imagen del firmware en la SRAM es una operación más rápida porque lanza el archivo de ejecución inmediatamente. Este método se utiliza principalmente para el desarrollo iterativo.

   Utilice los siguientes comandos para cargar el firmware en la SRAM.

   ```
   cd amzsdk_bundle-x.y.z
   ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/ramload.py
    build/cmake/vendors/marvell/mw300_rd/aws_demos.axf
   ```

   La salida del comando se muestra en la figura siguiente.

```
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
 select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
75072 bytes written at address 0x00100000
8 bytes written at address 0x00112540
468 bytes written at address 0x20000040
downloaded 75548 bytes in 0.636127s (115.979 KiB/s)
verified 75548 bytes in 0.959023s (76.930 KiB/s)
shutdown command invoked
```

Cuando finalice la ejecución del comando, debería ver los registros de la aplicación de demostración.

Creación y ejecución de la demostración de FreeRTOS con el IDE de Eclipse

1. Antes de configurar un espacio de trabajo de Eclipse, debe ejecutar el comando `cmake`.

   Ejecute el siguiente comando para trabajar con el proyecto de Eclipse `aws_demos`.

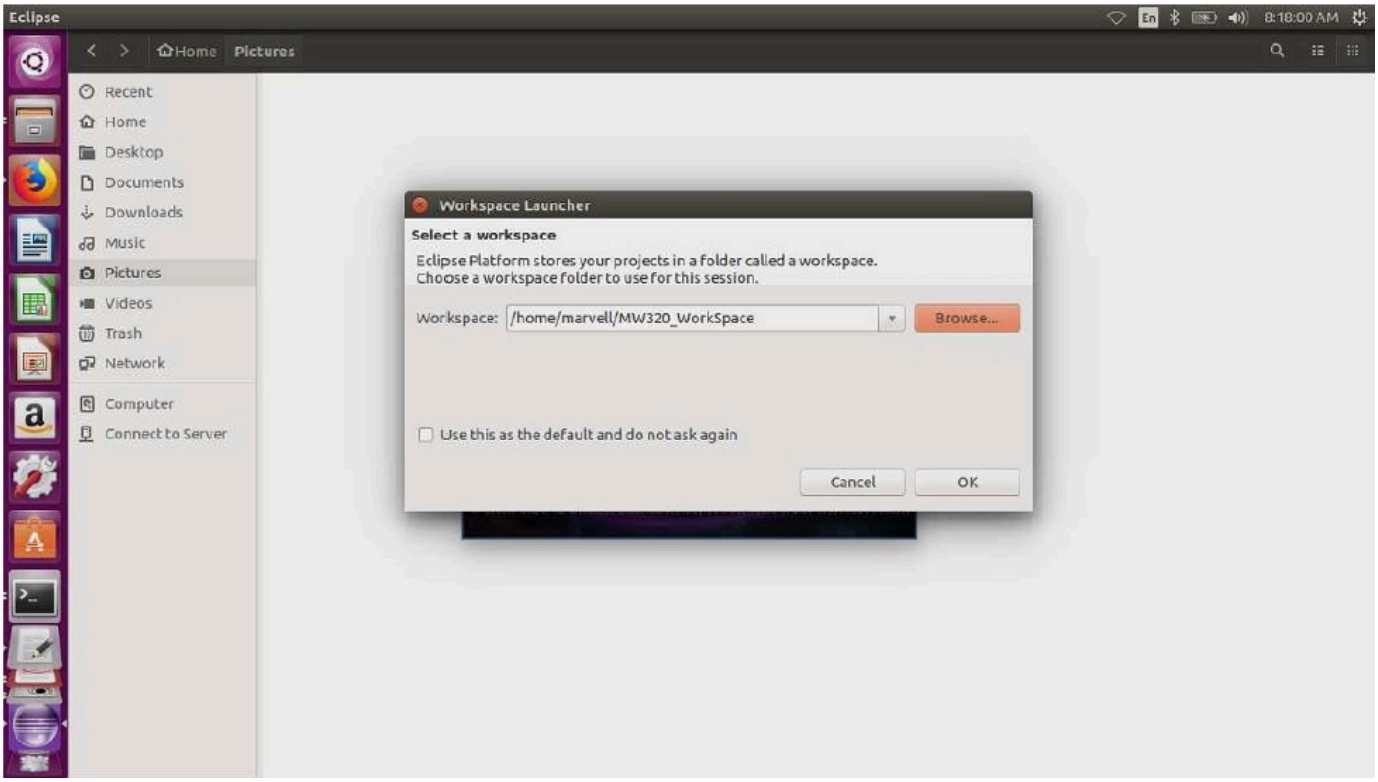
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Ejecute el siguiente comando para trabajar con el proyecto de Eclipse `aws_tests`.

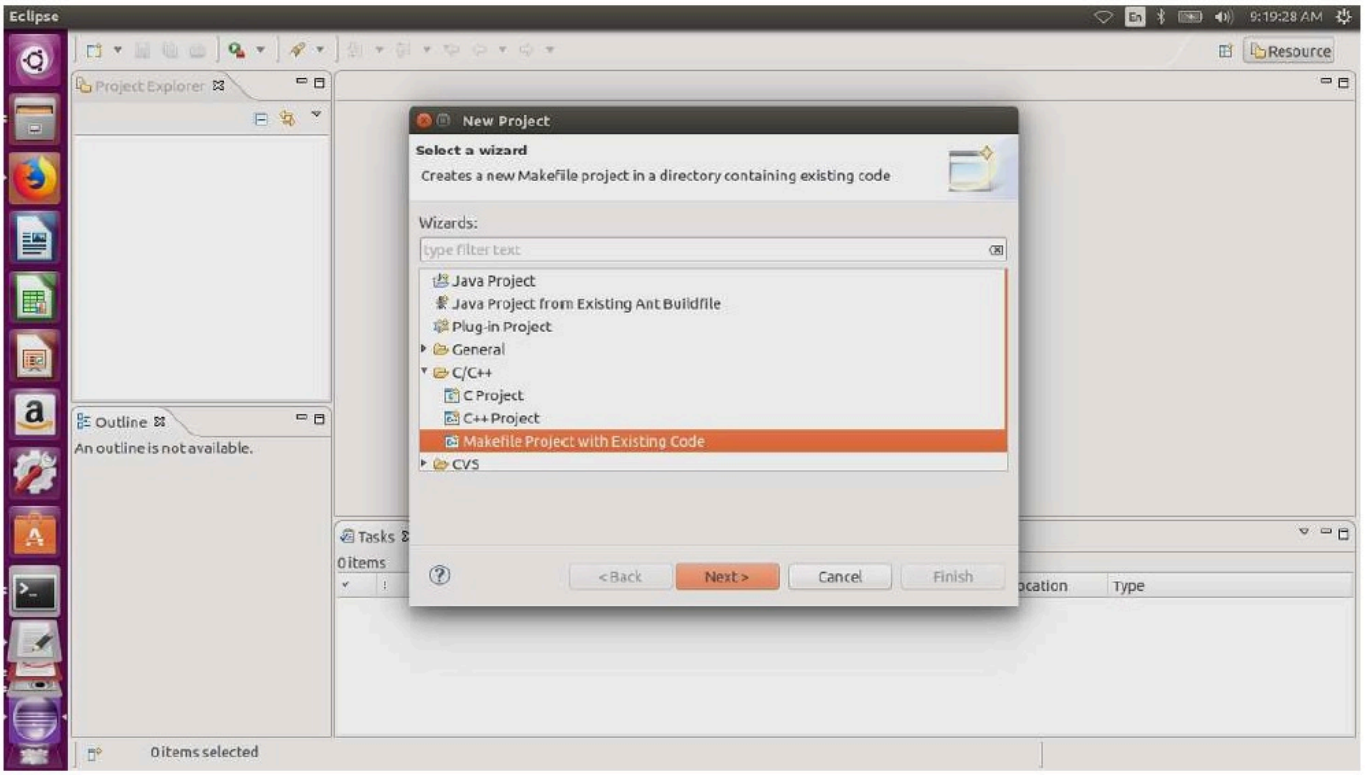
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> Ejecute el comando `cmake` cada vez que cambie entre el proyecto `aws_demos` y el proeycto `aws_tests`.

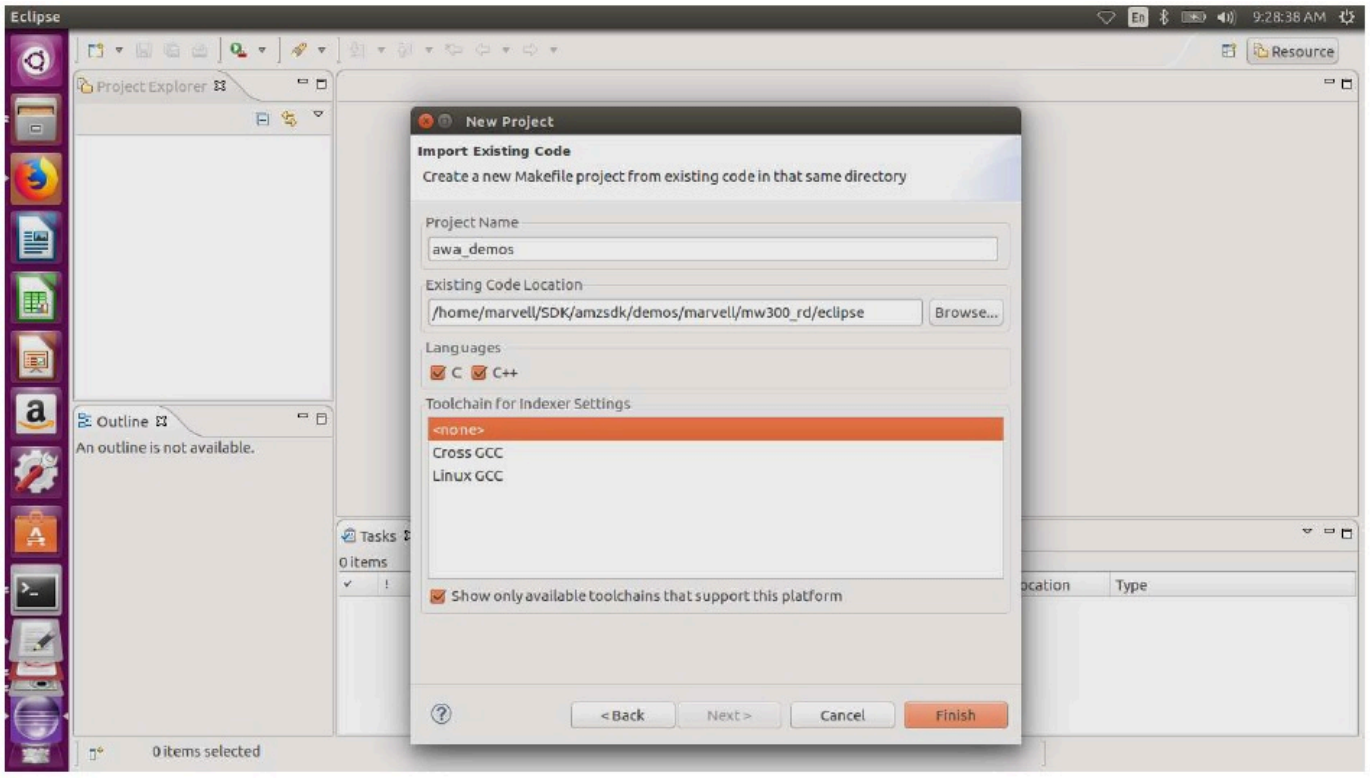
2. Abre Eclipse y, cuando se le pida, elija su espacio de trabajo de Eclipse como se muestra en la siguiente figura.



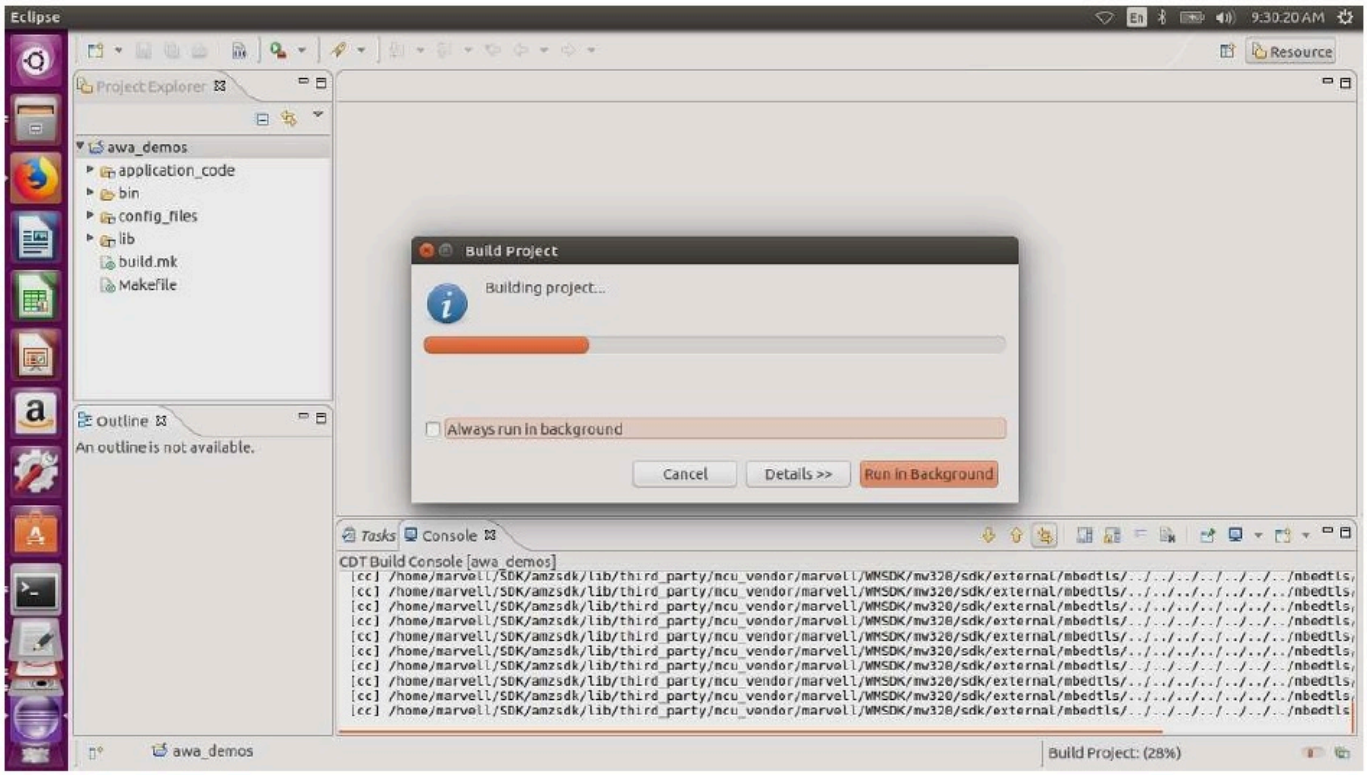
3. Elija la opción para crear un Proyecto de Makefile: con el código existente, como se muestra en la siguiente figura.



4. Elija Examinar, especifique el directorio del código existente y, a continuación, elija Finalizar.



5. En el panel de navegación, elija aws_demos en el explorador de proyectos. Haga clic con el botón derecho en aws_demos para abrir el menú y, a continuación, seleccione Crear.



Si la creación se realiza correctamente, se genera el archivo `build/cmake/vendors/marvell/mw300_rd/aws_demos.bin`.

6. Utilice las herramientas de línea de comandos para actualizar el archivo de diseño (`layout.txt`), el binario Boot2 (`boot2.bin`), el binario del firmware de la MCU (`aws_demos.bin`) y el firmware de Wi-Fi.

   a. Antes de instalar la imagen del firmware, prepare la memoria flash de la placa de desarrollo con los componentes comunes Layout y Boot2. Use los siguientes comandos.

   ```
   cd amzsdk_bundle-x.y.z
   ./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/
   marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/
   marvell/WMSDK/mw320/boot2/bin/boot2.bin
   ```

   El comando `flashprog` inicia lo siguiente:

   - Diseño - Primero se indica a la utilidad flashprog que escriba un diseño en la memoria flash. El diseño es similar al de la información de partición de la memoria flash. El diseño predeterminado se encuentra en `/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.

- Boot2 - Este es el gestor de arranque utilizado por el WMSDK. El comando flashprog también escribe un cargador de arranque en la memoria flash. El cargador de arranque carga la imagen del firmware del microcontrolador después de que se haya instalado. Asegúrese de obtener la misma salida que la que se muestra en la siguiente figura.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245458s (115.728 KiB/s)
verified 29088 bytes in 0.350004s (81.160 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

b. El firmware utiliza el chipset Wi-Fi para su funcionalidad, y el chipset Wi-Fi tiene su propio firmware que también debe estar presente en la memoria flash. La utilidad `flashprog.py` se utiliza para instalar el firmware de Wi-Fi de la misma forma que se utilizó para actualizar el gestor de arranque boot2 y el firmware de la MCU. Utilice los siguientes comandos para instalar el firmware de Wi-Fi.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./
vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

Asegúrese de que el resultado del comando sea similar al de la siguiente figura.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary).....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

c. Utilice los siguientes comandos para instalar el firmware de la MCU.

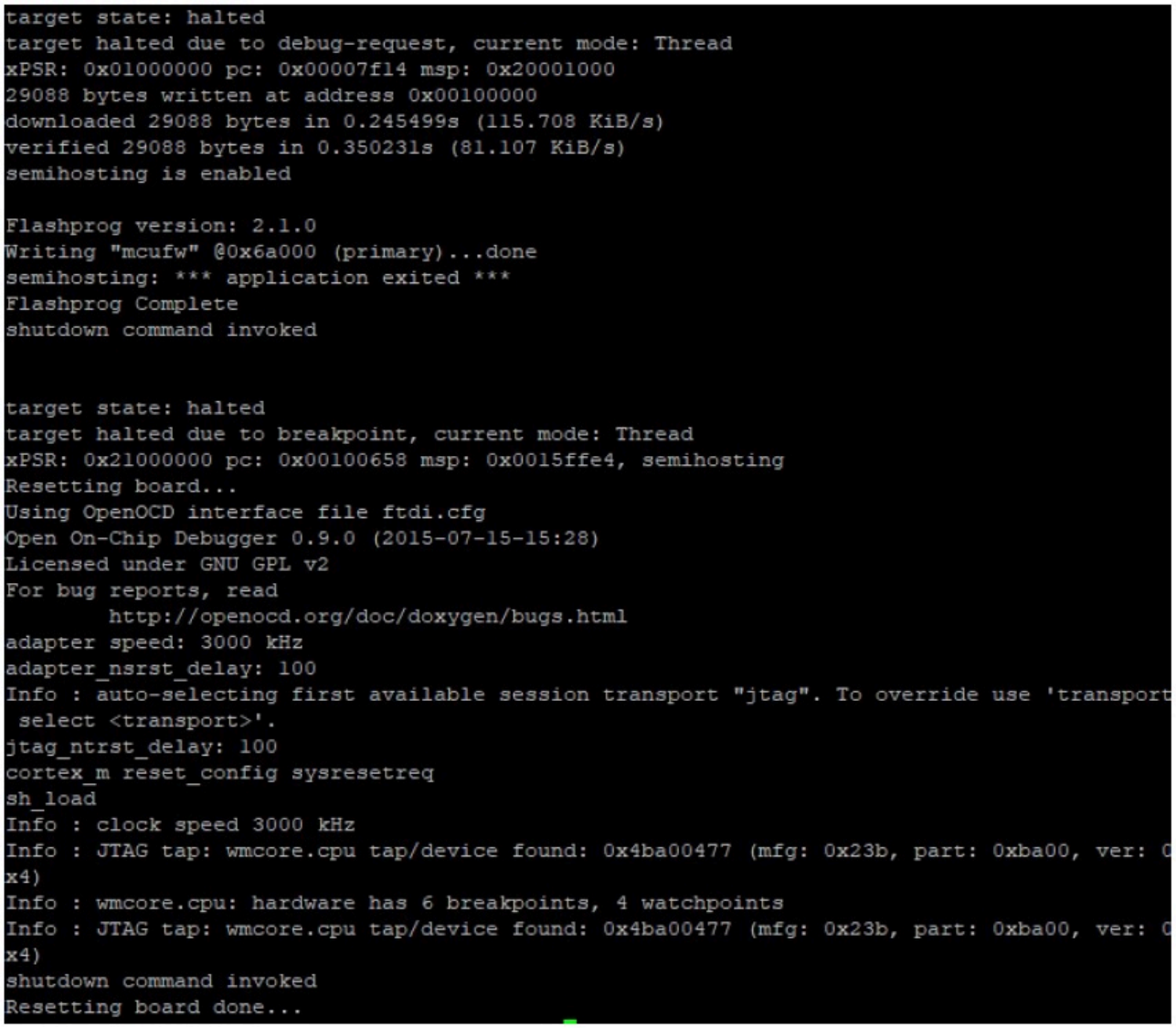
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. Restablezca la placa. Debería ver los registros de la aplicación de demostración.

e. Para ejecutar la aplicación de prueba, actualice el binario `aws_tests.bin` ubicado en el mismo directorio.

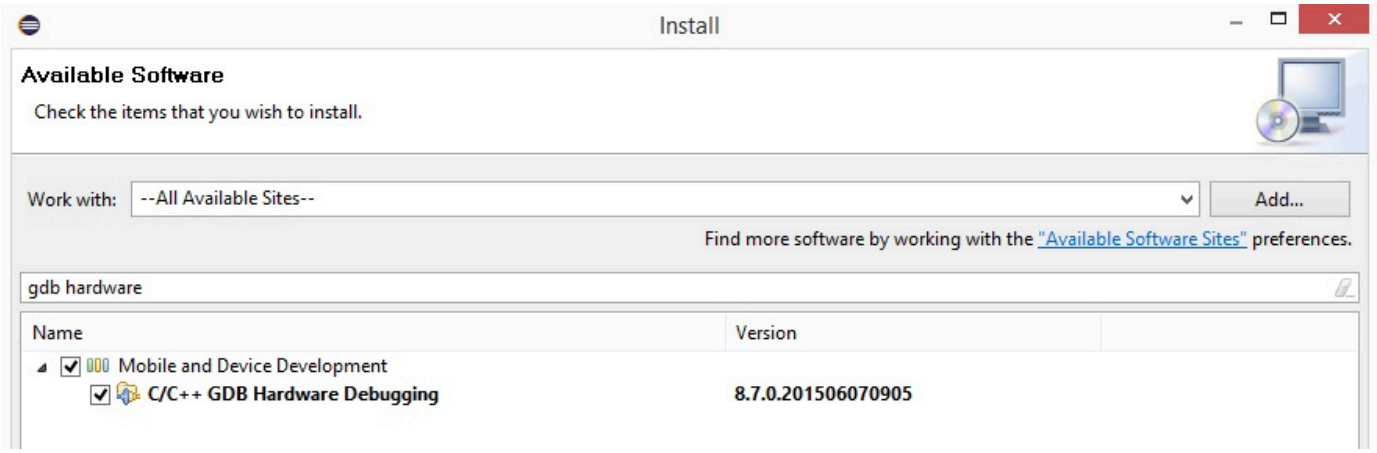
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

La salida del comando debería ser similar a la que se muestra en la figura siguiente.



Debugging

- Inicie Eclipse, seleccione Ayuda y, a continuación, elija Instalar software nuevo. En el menú Trabajar con, seleccione Todos los sitios disponibles. Introduzca el texto del filtro `GDB Hardware`. Seleccione la opción Depuración de hardware GDB en C/C++ e instale el complemento.



Solución de problemas

Problemas de red

Compruebe sus credenciales de red. Consulte “Aprovisionamiento” en Creación y ejecución del proyecto de demostración de FreeRTOS.

Habilitación de registros adicionales

- Habilite los registros específicos de la placa.

  Habilite las llamadas a `wmstdio_init(UART0_ID, 0)` en la función `prvMiscInitialization` del archivo `main.c` para realizar pruebas o demostraciones.

- Habilitación de los registros de Wi-Fi

  Habilite la macro `CONFIG_WLCMGR_DEBUG` en el archivo *`freertos`*`/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h`.

Uso de GDB

Le recomendamos que utilice los archivos de comandos `arm-none-eabi-gdb` y `gdb` empaquetados junto con el SDK. Vaya al directorio .

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

Ejecute el siguiente comando (en una sola línea) para conectarse a GDB.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/
vendors/marvell/mw300 _rd/aws_demos.axf
```

Introducción al kit de desarrollo MediaTek MT7697hx

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial proporciona instrucciones para empezar a utilizar el kit de desarrollo MT7697hx. MediaTek Si no tiene el kit de desarrollo MediaTek MT7697hx, visite el catálogo de dispositivos de nuestros socios para comprar uno de AWS nuestros socios.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *`freertos`*.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
2. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
3. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.
4. Interacción con la aplicación que se ejecuta en la placa con una conexión serie para fines de monitorización y depuración.

Configure el entorno de desarrollo.

Antes de configurar el entorno, conecte el ordenador al puerto USB del kit de desarrollo MT7697hx. MediaTek

Descarga e instalación de Keil MDK

Puede utilizar el Kit de desarrollo de microcontroladores (MDK) de Keil basado en GUI para configurar, crear y ejecutar proyectos de FreeRTOS en su placa. El MDK de Keil incluye el IDE de µVision y el depurador de µVision.

> **Note**
>
> Keil MDK solo es compatible con equipos Windows 7, Windows 8 y Windows 10 de 64 bits.

Para descargar e instalar Keil MDK

1. Vaya a la página de introducción de Keil MDK y elija Download MDK-Core (Descargar MDK-Core).
2. Especifique y envíe su información para registrarse en Keil.
3. Haga clic con el botón derecho en el archivo ejecutable de MDK y guarde el instalador de MDK Keil en su equipo.
4. Abra el instalador de Keil MDK y siga los pasos de instalación. Asegúrese de instalar el paquete de MediaTek dispositivos (serie MT76x7).

Establecimiento de una conexión serie

Conecte la placa al equipo host con un cable USB. Aparece un puerto COM en el Administrador de dispositivos de Windows. Para la depuración, puede abrir una sesión en el puerto con una herramienta de utilidad de terminal, como o. HyperTerminal TeraTerm

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje “¡Hola, mundo!” enviado varias veces al tema al que se ha suscrito.

Creación y ejecución del proyecto de demostración de FreeRTOS con el MDK de Keil

Para crear el proyecto de demostración de FreeRTOS en Keil µVision

1. En el menú Inicio, abra Keil µVision 5.
2. Abra el archivo de proyecto `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx`.
3. En el menú, elija Project (Proyecto) y después Build target (Compilar destino).

   Una vez compilado el código, puede ver el archivo ejecutable de la demostración en `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf`.

Para ejecutar el proyecto de demostración de FreeRTOS

1. Configure el kit de desarrollo MediaTek MT7697hx en modo PROGRAMAR.

   Para establecer el kit en el modo PROGRAM, mantenga pulsado el botón PROG. Con el botón PROG pulsado, pulse y suelte el botón RESET y después suelte el botón PROG.
2. En el menú, elija Flash y después Configure Flash Tools (Configurar herramientas Flash).
3. En Opciones para el destino “**`aws_demo`**”, seleccione la pestaña Depurar. Seleccione Use (Usar), establezca el depurador en CMSIS-DAP Debugger (Depurador CMSIS-DAP) y, a continuación, elija OK (Aceptar).
4. En el menú, elija Flash y, a continuación, elija Download (Descargar).

   µVision le avisa cuando se completa la descarga.
5. Utilice una utilidad de terminal para abrir la ventana de la consola de serie. Establezca el puerto serie en 115200 bps, ninguna paridad, 8 bits y 1 bit de parada.
6. Pulse el botón RESET en su MediaTek kit de desarrollo MT7697hx.

Solución de problemas

Depuración de proyectos de FreeRTOS en Keil µVision

Actualmente, debe editar el MediaTek paquete que se incluye con Keil µVision antes de poder depurar el proyecto de demostración de FreeRTOS con Keil µVision. MediaTek

Para editar el MediaTek paquete para depurar proyectos de FreeRTOS

1. Busque y abra el archivo `Keil_v5\ARM\PACK\.Web\MediaTek.MTx.pdsc` en la carpeta de instalación de Keil MDK.
2. Reemplace todas las instancias de `flag = Read32(0x20000000);` por `flag = Read32(0x0010FBFC);`.
3. Reemplace todas las instancias de `Write32(0x20000000, 0x76877697);` por `Write32(0x0010FBFC, 0x76877697);`.

Para empezar a depurar el proyecto

1. En el menú, elija Flash y después Configure Flash Tools (Configurar herramientas Flash).
2. Elija la pestaña Target (Destino) y después elija Read/Write Memory Areas (Áreas de memoria de lectura/escritura). Confirme que IRAM1 y IRAM2 están seleccionados.
3. Seleccione la pestaña Debug (Depurar) y, a continuación, seleccione CMSIS-DAP Debugger (Depurador de CMSIS-DAP).
4. Abra `vendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c` y establezca la macro `MTK_DEBUGGER` en `1`.
5. Vuelva a crear el proyecto de demostración en µVision.
6. Configure el kit de MediaTek desarrollo MT7697Hx en modo PROGRAMA.

   Para establecer el kit en el modo PROGRAM, mantenga pulsado el botón PROG. Con el botón PROG pulsado, pulse y suelte el botón RESET y después suelte el botón PROG.
7. En el menú, elija Flash y, a continuación, elija Download (Descargar).

   µVision le avisa cuando se completa la descarga.
8. Pulse el botón RESET del kit de desarrollo del MT7697hx MediaTek .
9. En el menú de µVision, elija Depurar y después Iniciar/detener sesión de depuración. La ventana Call Stack + Locals (Pila de llamadas + Variables locales) se abre cuando inicia la sesión de depuración.
10. En el menú, elija Debug (Depurar) y, a continuación, elija Stop (Detener) para detener la ejecución del código. El contador del programa se detiene en la siguiente línea:

```
{ volatile int wait_ice = 1 ; while ( wait_ice ) ; }
```

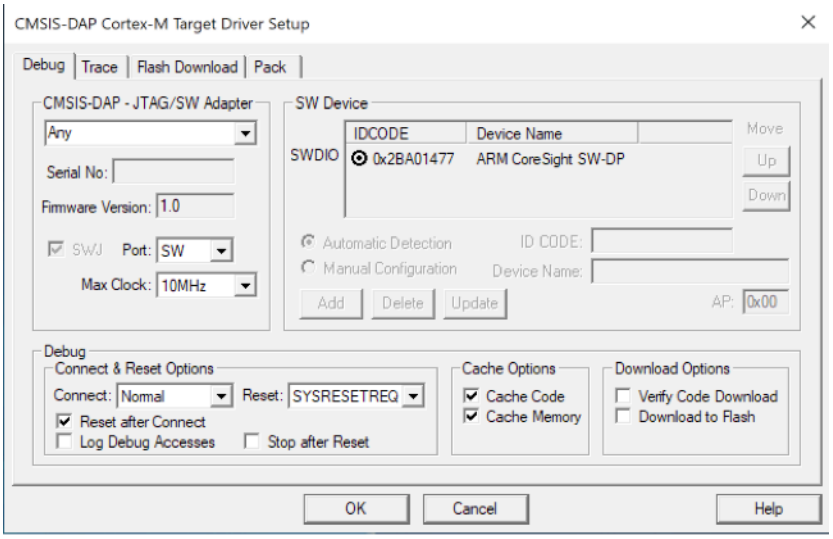
11. En la ventana Call Stack + Locals (Pila de llamadas + Variables locales), cambie el valor de `wait_ice` a `0`.
12. Defina puntos de interrupción en el código fuente del proyecto y ejecute el código.

Solución de problemas de la configuración del depurador de IDE.

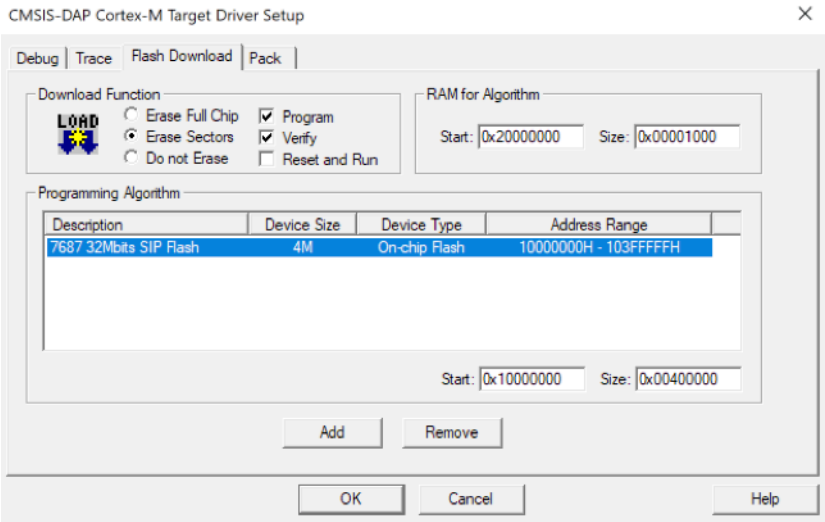
Si tiene problemas con la depuración de una aplicación, puede que la configuración de su depurador no sea correcta.

Para verificar que la configuración de su depurador es correcta

1. Abra Keil µVision.
2. Haga clic con el botón derecho en el proyecto de `aws_demos`, elija Options (Opciones) y en la pestaña Utilities (Utilidades), elija Settings (Configuración), situado junto a “-- Use Debug Driver --” (Utilizar controlador del depurador).
3. Compruebe que la configuración de la pestaña Debug (Depurador) aparece de la siguiente manera:



4. Compruebe que la configuración de la pestaña Flash Download (Descarga flash) aparece de la siguiente manera:



Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción a Microchip Curiosity PIC32MZ EF

Important

Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Note

De acuerdo con Microchip, eliminaremos Curiosity PIC32MZEF (DM320104) de la rama principal del repositorio de integración de referencias de FreeRTOS y ya no lo incluiremos en las nuevas versiones. Microchip ha publicado un aviso oficial en el que indica que PIC32MZEF (DM320104) ya no se recomienda para los nuevos diseños. Aún se puede acceder a los proyectos y al código fuente de PIC32MZEF a través de las etiquetas de las versiones anteriores. Microchip recomienda a los clientes que utilicen la placa de desarrollo PIC32MZ-EF-2.0 (DM320209) de Curiosity para los nuevos diseños. La plataforma PIC32MZv1 todavía se encuentra en la versión 202012.00 del repositorio de integración de

referencias de FreeRTOS. Sin embargo, la plataforma ya no es compatible con la versión 202107.00 de la referencia de FreeRTOS.

Este tutorial ofrece instrucciones para la introducción a Microchip Curiosity PIC32MZ EF Si no tiene el paquete Microchip Curiosity PIC32MZ EF, consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestro socio.

El paquete incluye los elementos siguientes:

- Placa de desarrollo Curiosity PIC32MZ EF
- Placa acoplable MikroElectronika USB UART
- Placa acoplable MikroElectronika WiFi 7
- Placa secundaria PIC32 LAN8720 PHY

También necesita los siguientes elementos para la depuración:

- Depurador en circuito MPLAB Snap
- (Opcional) Kit de cable de programación PICkit 3

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos.

⚠ Important

- En este tema, la ruta al directorio de descargas de FreeRTOS se denomina *`freertos`*.
- Los caracteres de espacio en la ruta *`freertos`* pueden causar errores de compilación. Al clonar o copiar el repositorio, asegúrese de que la ruta que crea no contiene caracteres de espacio.
- La longitud máxima de una ruta de archivo en Microsoft Windows es de 260 caracteres. Las rutas largas al directorio de descargas de FreeRTOS pueden provocar errores de creación.
- Como el código fuente puede contener enlaces simbólicos, si utiliza Windows para extraer el archivo, es posible que tenga que:
  - Habilitar el modo de desarrollador o

- Utilizar una consola que tenga el rango de administrador.

De esta forma, Windows puede crear correctamente enlaces simbólicos al extraer el archivo. De lo contrario, los enlaces simbólicos se escribirán como archivos normales que contengan las rutas de los enlaces simbólicos como texto o estarán vacíos. Para obtener más información, consulte la entrada del blog Symlinks in Windows 10 .

Si usa Git en Windows, debe habilitar el modo desarrollador o debe:

- Establecer `core.symlinks` en verdadero con el siguiente comando:

```
git config --global core.symlinks true
```

- Usar una consola que tenga el rango de administrador siempre que utilice un comando git que escriba en el sistema (por ejemplogit pull, git clone y git submodule update --init --recursive).

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Conexión de su placa a un equipo host.
2. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
3. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
4. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.
5. Interacción con la aplicación que se ejecuta en la placa con una conexión serie para fines de monitorización y depuración.

Configure el hardware de Microchip Curiosity PIC32MZ EF

1. Conecte la placa acoplable MikroElectronika USB UART al conector del microBUS 1 en Microchip Curiosity PIC32MZ EF.
2. Conecte la placa secundaria PIC32 LAN8720 PHY al encabezado J18 en Microchip Curiosity PIC32MZ EF.

3. Conecte la placa acoplable MikroElectronika USB UART a su equipo mediante un cable USB A a USB mini-B.
4. Para conectar su placa a Internet, utilice una de las siguientes opciones:

   - Para utilizar wifi, conecte la placa acoplable MikroElectronika WiFi 7 al conector del microBUS 2 en Microchip Curiosity PIC32MZ EF. Consulte Configuración de las demostraciones de FreeRTOS.
   - Para utilizar Ethernet para conectar la placa Microchip Curiosity PIC32MZ EF a Internet, conecte la placa secundaria PIC32 LAN8720 PHY al encabezado J18 del microchip Curiosity PIC32MZ EF. Conecte un extremo de un cable Ethernet a la placa secundaria LAN8720 PHY. Conecte el otro extremo a su router u otro puerto de Internet. También debe definir la macro del preprocesador `PIC32_USE_ETHERNET`.
5. Si no lo ha hecho ya, suelde el conector en ángulo al encabezado ICSP en Microchip Curiosity PIC32MZ EF.
6. Conecte un extremo del cable ICSP desde el kit de cable de programación PICkit 3 a Microchip Curiosity PIC32MZ EF.

   Si no tiene el kit de cable de programación PICkit 3, puede usar los puentes de cables M-F Dupont para establecer la conexión. Tenga en cuenta que el círculo blanco significa la posición de la clavija 1.
7. Conecte el otro extremo del cable ICSP (o puentes) al depurador de ajuste MPLAB. La clavija 1 del conector de programación SIL de 8 clavijas está marcada por el triángulo negro en la parte inferior derecha de la placa.

   Asegúrese de que cualquier cableado a la clavija 1 de Microchip Curiosity PIC32MZ EF, indicado por el círculo blanco, esté alineado con la clavija 1 del depurador de ajuste MPLAB.

   Para obtener más información sobre el depurador MPLAB Snap, consulte la Hoja de información del depurador en circuito MPLAB Snap.

Configure el hardware del microchip Curiosity PIC32MZ EF con PICkit On Board (PKOB)

Le recomendamos que siga el procedimiento de configuración de la sección anterior. Sin embargo, puede evaluar y ejecutar demostraciones de FreeRTOS con la depuración básica utilizando el programador/depurador integrado PICkit On Board (PKOB) siguiendo estos pasos.

1. Conecte la placa acoplable MikroElectronika USB UART al conector del microBUS 1 en Microchip Curiosity PIC32MZ EF.
2. Para conectar la placa a Internet, realice una de las siguientes acciones:
   - Para utilizar wifi, conecte la placa acoplable MikroElectronika WiFi 7 al conector del microBUS 2 en Microchip Curiosity PIC32MZ EF. (Siga los pasos “Para configurar la wi-fi” en Configuración de las demostraciones de FreeRTOS.
   - Para utilizar Ethernet para conectar la placa Microchip Curiosity PIC32MZ EF a Internet, conecte la placa secundaria PIC32 LAN8720 PHY al encabezado J18 del microchip Curiosity PIC32MZ EF. Conecte un extremo de un cable Ethernet a la placa secundaria LAN8720 PHY. Conecte el otro extremo a su router u otro puerto de Internet. También debe definir la macro del preprocesador `PIC32_USE_ETHERNET`.
3. Conecte el puerto USB micro-B denominado “USB DEBUG” de la placa del microchip Curiosity PIC32MZ EF a su equipo utilizando un cable USB tipo A a USB micro-B.
4. Conecte la placa acoplable MikroElectronika USB UART a su equipo mediante un cable USB A a USB mini-B.

Configure el entorno de desarrollo.

> **Note**
>
> El proyecto de FreeRTOS para este dispositivo se basa en MPLAB Harmony v2. Para compilar el proyecto, tendrá que utilizar versiones de las herramientas de MPLAB compatibles con Harmony v2, como la v2.10 del compilador MPLAB XC32 y las versiones 2.X.X del MPLAB Harmony Configurator (MHC).

1. Instale Python versión 3.x o una versión posterior.
2. Instale el IDE de MPLAB X:

   > **Note**
   >
   > Actualmente, Integraciones de referencia de AWS v202007.00 de FreeRTOS solo es compatible con MPLAB v5.35. Las versiones anteriores de Integraciones de referencia de AWS de Freertos son compatibles con MPLabv5.40.

Descargas de MPLabv5.35

- Entorno de desarrollo integrado de MPLAB X para Windows
- Entorno de desarrollo integrado de MPLAB X para macOS
- Entorno de desarrollo integrado de MPLAB X para Linux

Descargas más recientes de MPlab (MPLabv5.40)

- Entorno de desarrollo integrado (IDE) de MPLAB X para Windows
- Entorno de desarrollo integrado (IDE) de MPLAB X para macOS
- Entorno de desarrollo integrado (IDE) de MPLAB X para Linux

3. Instale el compilador MPLAB XC32:

   - Compilador MPLAB XC32/32++ para Windows
   - Compilador MPLAB XC32/32++ para macOS
   - Compilador MPLAB XC32/32++ para Linux

4. Inicie un emulador de terminal UART y abra una conexión con los siguientes valores de configuración:

   - Velocidad en baudios: 115 200
   - Datos: 8 bits
   - Paridad: ninguna
   - Bits de parada: 1
   - Control del flujo: ninguno

Supervisión de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.

2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje "¡Hola, mundo!" enviado varias veces al tema al que se ha suscrito.

Creación y ejecución del proyecto de demostración de FreeRTOS

Apertura de la demostración de FreeRTOS en el IDE de MPLAB

1. Abra el IDE de MPLAB. Si tiene más de una versión del compilador instalada, debe seleccionar el compilador que desea utilizar desde el IDE.
2. En el menú File (Archivo), elija Open project (Abrir proyecto).
3. Vaya a `projects/microchip/curiosity_pic32mzef/mplab/aws_demos` y abra.
4. Elija Open project (Abrir proyecto).

> **Note**
>
> Al abrir el proyecto por primera vez, es posible que aparezca un mensaje de error sobre el compilador. En el IDE, vaya a Tools (Herramientas), Options (Opciones), Embedded (Incrustado) y seleccione el compilador que utiliza para su proyecto.

Para utilizar Ethernet para conectarse, debe definir la macro del preprocesador `PIC32_USE_ETHERNET`.

Para usar Ethernet para conectarse mediante el IDE de MPLAB

1. En el IDE de MPLAB, haga clic con el botón derecho en el proyecto y, a continuación, elija Propiedades.
2. En el cuadro de diálogo Propiedades del proyecto, elija ***compiler-name*** (Opciones globales) para expandirlo y, a continuación, seleccione ***compiler-name***-gcc.
3. En Categorías de opciones, elija Preprocesamiento y mensajes y, a continuación, añada la cadena `PIC32_USE_ETHERNET` a las macros del preprocesador.

Ejecución del proyecto de demostración de FreeRTOS

1. Vuelva a compilar el proyecto.
2. En la pestaña Projects (Proyectos), haga clic con el botón derecho del ratón en la carpeta de nivel superior `aws_demos` y, a continuación, elija Debug (Depurar).
3. Cuando el depurador se detenga en el punto de ruptura en `main()`, desde el menú Run (Ejecutar), elija Resume (Reanudar).

Creación de la demostración de FreeRTOS con CMake

Si prefiere no utilizar un IDE para el desarrollo de FreeRTOS, también puede usar CMake para crear y ejecutar las aplicaciones de demostración o las aplicaciones que ha desarrollado con herramientas de depuración y editores de código de terceros.

Para crear la demostración de FreeRTOS con CMake

1. Cree un directorio que contenga los archivos de creación generados, como *`build-directory`*.
2. Utilice el siguiente comando para generar los archivos de creación del código fuente.

```
cmake -DVENDOR=microchip -DBOARD=curiosity_pic32mzef -DCOMPILER=xc32 -
DMCHP_HEXMATE_PATH=path/microchip/mplabx/version/mplab_platform/bin  -
DAFR_TOOLCHAIN_PATH=path/microchip/xc32/version/bin -S freertos -B build-folder
```

> **Note**
>
> Debe especificar las rutas correctas a los binarios de la cadena de herramientas y Hexmate, como `C:\Program Files (x86)\Microchip\MPLABX \v5.35\mplab_platform\bin` y `C:\Program Files\Microchip \xc32\v2.40\bin`.

3. Cambie los directorios al directorio de creación (*`build-directory`*) y ejecute `make` desde dicho directorio.

Para obtener más información, consulte Uso de CMake con FreeRTOS.

Para utilizar Ethernet para conectarse, debe definir la macro del preprocesador `PIC32_USE_ETHERNET`.

Solución de problemas

Para obtener información sobre la resolución de problemas, consulte Introducción a solución de problemas.

Introducción a Nordic nRF52840-DK

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción a Nordic nRF52840-DK. Si no tiene el Nordic nRF52840-DK, consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestro socio.

Antes de comenzar, necesita una Configuración de AWS IoT y Amazon Cognito para Bluetooth de bajo consumo de FreeRTOS.

Para ejecutar la demostración de Bluetooth de bajo consumo de FreeRTOS, también necesita un dispositivo móvil iOS o Android con funciones de Bluetooth y Wi-Fi.

> **Note**
>
> Si está utilizando un dispositivo iOS, necesita Xcode para crear la aplicación móvil de demostración. Si está utilizando un dispositivo Android, puede utilizar Android Studio para crear la aplicación móvil de demostración.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Conexión de su placa a un equipo host.
2. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.

3. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
4. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.
5. Interacción con la aplicación que se ejecuta en la placa con una conexión serie para fines de monitorización y depuración.

Configuración del hardware Nordic

Conecte el equipo host al puerto USB con la etiqueta J2, que se encuentra directamente encima del soporte de la pila de botón en su placa Nordic nRF52840.

Para obtener más información acerca de cómo configurar la Nordic nRF52840-DK, consulte nRF52840 Development Kit User Guide.

Configure el entorno de desarrollo.

Descargar e instalar Segger Embedded Studio

FreeRTOS admite Segger Embedded Studio como un entorno de desarrollo para Nordic nRF52840-DK.

Para configurar su entorno, tendrá que descargar e instalar Segger Embedded Studio en su equipo host.

Para descargar e instalar Segger Embedded Studio

1. Vaya a la página de descargas de Segger Embedded Studio y elija la opción Embedded Studio for ARM para su sistema operativo.
2. Ejecute el instalador y siga las instrucciones para finalizar.

Configuración de la aplicación de demostración del SDK para móviles de Bluetooth de bajo consumo de FreeRTOS

Para ejecutar el proyecto de demostración de FreeRTOS con Bluetooth de bajo consumo, debe ejecutar la aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS en su dispositivo móvil.

Para configurar la aplicación de demostración del SDK para móviles de Bluetooth de bajo consumo de FreeRTOS

1. Siga las instrucciones de SDK para móviles para dispositivos Bluetooth de FreeRTOS para descargar e instalar los SDK para su plataforma móvil en su equipo host.
2. Siga las instrucciones en Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS para configurar la aplicación móvil de demostración en su dispositivo móvil.

Establecimiento de una conexión serie

Segger Embedded Studio incluye un emulador de terminal que puede utilizar para recibir mensajes de registro a través de una conexión en serie en su placa.

Para establecer una conexión en serie con Segger Embedded Studio

1. Abra Segger Embedded Studio.
2. En el menú superior, seleccione Target (Destino), Connect J-Link (Conectar J-Link).
3. En el menú superior, elija Tools (Herramientas), Terminal Emulator (Emulador de terminal), Properties (Propiedades) y defina las propiedades según se indica en Instalación de un emulador de terminal.
4. En el menú superior, elija Tools (Herramientas), Terminal Emulator (Emulador de terminal), Connect ***puerto*** (115200,N,8,1) (Conectar puerto (115200,N,8,1)).

> **Note**
>
> El emulador de terminal de estudio integrado Segger no admite una funcionalidad de entrada. Para esto, use un emulador de terminal como PuTTy, Tera Term o GNU Screen. Configure el terminal para que se conecte a la placa mediante una conexión en serie, tal como se indica en Instalación de un emulador de terminal.

Descarga y configuración de FreeRTOS

Después de configurar el hardware y el entorno, puede descargar FreeRTOS.

Descarga de FreeRTOS

Para descargar FreeRTOS para Nordic nRF52840-DK, vaya a la página de GitHub de FreeRTOS y clone el repositorio. Consulte el archivo README.md para obtener instrucciones.

Important

- En este tema, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.
- Los caracteres de espacio en la ruta *freertos* pueden causar errores de compilación. Al clonar o copiar el repositorio, asegúrese de que la ruta que crea no contiene caracteres de espacio.
- La longitud máxima de una ruta de archivo en Microsoft Windows es de 260 caracteres. Las rutas largas al directorio de descargas de FreeRTOS pueden provocar errores de creación.
- Como el código fuente puede contener enlaces simbólicos, si utiliza Windows para extraer el archivo, es posible que tenga que:
  - Habilitar el modo de desarrollador o
  - Utilizar una consola que tenga el rango de administrador.

  De esta forma, Windows puede crear correctamente enlaces simbólicos al extraer el archivo. De lo contrario, los enlaces simbólicos se escribirán como archivos normales que contengan las rutas de los enlaces simbólicos como texto o estarán vacíos. Para obtener más información, consulte la entrada del blog Symlinks in Windows 10 .

  Si usa Git en Windows, debe habilitar el modo desarrollador o debe:

  - Establecer `core.symlinks` en verdadero con el siguiente comando:

    ```
    git config --global core.symlinks true
    ```

  - Usar una consola que tenga el rango de administrador siempre que utilice un comando git que escriba en el sistema (por ejemplogit pull, git clone y git submodule update --init --recursive).

Configure su proyecto

Para habilitar la demostración, tiene que configurar su proyecto para que funcione con AWS IoT. Para configurar el proyecto para trabajar con AWS IoT, el dispositivo debe registrarse como un objeto de AWS IoT. Debe haber registrado su dispositivo al Configuración de AWS IoT y Amazon Cognito para Bluetooth de bajo consumo de FreeRTOS.

Para configurar su punto de enlace de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Settings (Configuración).

   El punto de conexión de AWS IoT aparece en el cuadro de texto Punto de enlace de datos de dispositivo. Debe tener un aspecto similar al siguiente: *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Tome nota de este punto de enlace.
3. En el panel de navegación, elija Manage (Administrar) y, a continuación, Things (Cosas). Anote el nombre de objeto de AWS IoT de su dispositivo.
4. Con el punto de enlace de AWS IoT y el nombre de objeto de AWS IoT a mano, abra *freertos*`/demos/include/aws_clientcredential.h` en el IDE y especifique los valores para las siguientes constantes `#define`:

   - `clientcredentialMQTT_BROKER_ENDPOINT` *Su punto de enlace de AWS IoT*
   - `clientcredentialIOT_THING_NAME` *Su nombre de objeto de AWS IoT*

Para habilitar la demostración

1. Compruebe que la demostración de GATT de Bluetooth de bajo consumo está habilitada. Vaya a `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/iot_ble_config.h` y añada `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` a la lista de instrucciones define.
2. Abra `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h` y defina `CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED` o `CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED` como en este ejemplo.

```
/* To run a particular demo you need to define one of these.
 * Only one demo can be configured at a time
 *
 *          CONFIG_BLE_GATT_SERVER_DEMO_ENABLED
```

```
 *          CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_SHADOW_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_POSIX_DEMO_ENABLED
 *
 *  These defines are used in iot_demo_runner.h for demo selection */

#define CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
```

3. Dado que el chip Nordic incluye muy poca RAM (250 KB), es posible que deba cambiarse la configuración de BLE para permitir el uso de entradas de tabla GATT de mayor tamaño en comparación con el tamaño de cada atributo. De esta manera puede ajustar la cantidad de memoria que recibe la aplicación. Para ello, reemplace las definiciones de los siguientes atributos en el archivo *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h`:

   - NRF_SDH_BLE_VS_UUID_COUNT

     El número de UUID específicos del proveedor. Aumente este recuento en 1 cuando añada un nuevo UUID específico del proveedor.

   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

     Tamaño de la tabla de atributos en bytes. El tamaño debe ser un múltiplo de 4. Este valor indica la cantidad fija de memoria dedicada a la tabla de atributos (incluido el tamaño de la característica), por lo que variará de un proyecto a otro. Si supera el tamaño de la tabla de atributos, aparecerá un error NRF_ERROR_NO_MEM. Si modifica NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE, normalmente también tendrá que volver a configurar los ajustes de la RAM.

   (Para las pruebas, la ubicación del archivo es *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h`.)

Creación y ejecución del proyecto de demostración de FreeRTOS

Una vez que haya descargado FreeRTOS y configurado su proyecto de demostración, estará listo para crear y ejecutar el proyecto de demostración en la placa.

> **Important**
>
> Si es la primera vez que se ejecute la demostración en esta placa, necesita instalar un cargador de arranque en la placa antes de poder ejecutar la demostración.
> Para compilar e instalar el cargador de arranque, siga los pasos que se indican a continuación, pero en lugar de usar el archivo de proyecto `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`, use `projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject`.

Para crear y ejecutar la demostración de Bluetooth de bajo consumo de FreeRTOS desde Segger Embedded Studio

1. Abra Segger Embedded Studio. Desde el menú superior, elija File (Archivo), elija Open Solution (Abrir solución) y, a continuación, desplácese hasta el archivo de proyecto `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`
2. Si utiliza el emulador de terminal de Segger Embedded Studio, elija Tools (Herramientas) en el menú superior y después elija Terminal Emulator (Emulador de terminal), Terminal Emulator (Emulador de terminal) para mostrar la información de su conexión en serie.

   Si utiliza otra herramienta de terminal, puede monitorizar esa herramienta para ver la salida de su conexión en serie.
3. Haga clic con el botón derecho en el proyecto de demostración `aws_demos` en el Project Explorer (Explorador de proyectos) y elija Build (Compilar).

   > **Note**
   >
   > Si es la primera vez que utiliza Segger Embedded Studio, es posible que vea una advertencia "No license for commercial use" (Sin licencia para uso comercial). Segger Embedded Studio se puede utilizar de forma gratuita para dispositivos semiconductores Nordic. Solicite una licencia gratuita y, durante la configuración, seleccione Active su licencia gratuita y siga las instrucciones.
4. Elija Debug (Depurar) y después elija Go (Ir).

   Después de que se inicie la demostración, espera a emparejarse con un dispositivo móvil mediante Bluetooth de bajo consumo.

5. Siga las instrucciones de la aplicación de demostración de MQTT a través de Bluetooth de bajo consumo para realizar la demostración con la aplicación de demostración del SDK para móviles de Bluetooth de bajo consumo de FreeRTOS como el proxy de MQTT para móviles.

Solución de problemas

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción a Nuvoton NuMaker-IoT-M487

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción a la placa de desarrollo Nuvoton NuMaker-IoT-M487. El microcontrolador de la serie incluye módulos RJ45 Ethernet y Wi-Fi integrados. Si no tiene una Nuvoton NuMaker-IoT-M487, consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestro socio.

Antes de comenzar, debe configurar AWS IoT y el software FreeRTOS para conectar la placa de desarrollo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *`freertos`*.

Información general

Este tutorial le guiará a través de los siguientes pasos:

1. Instale el software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
2. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria
3. Cargue la imagen binaria de la aplicación en su placa y, a continuación, ejecute la aplicación.

Configure el entorno de desarrollo.

La edición Keil MDK Nuvoton está diseñada para el desarrollo y la depuración de aplicaciones para las placas Nuvoton M487. La versión de Keil MDK v5 Essential, Plus o Pro también debe funcionar para la MCU Nuvoton M487 (núcleo Cortex-M4). Puede descargar la edición Keil MDK Nuvoton con un descuento en el precio de las MCU de la serie Cortex-M4 de Nuvoton. Keil MDK solo es compatible con Windows.

Para instalar la herramienta de desarrollo para NuMaker-IoT-M487

1. Descargue Keil MDK Nuvoton Edition desde el sitio web de Keil MDK.
2. Instale Keil MDK en su equipo host con su licencia. Keil MDK incluye el IDE de Keil µVision, una cadena de herramientas de compilación C/C++ y el depurador de µVision.

   Si tiene problemas durante la instalación, póngase en contacto con Nuvoton para obtener ayuda.
3. Instale Nu-Link_Keil_Driver_V3.06.7215r (o una versión más reciente), disponible en la página Herramienta de desarrollo de Nuvoton.

Creación y ejecución del proyecto de demostración de FreeRTOS

Para crear el proyecto de demostración de FreeRTOS

1. Abra el IDE de Keil µVision.
2. En el menú File (Archivo), elija Open (Abrir). En el cuadro de diálogo Open file (Abrir archivo), asegúrese de que el selector de tipo de archivo esté definido en Project Files (Archivos de proyecto).
3. Elija el proyecto de demostración de Wi-Fi o Ethernet que desea crear.

   - Para abrir el proyecto de demostración de wifi, elija el proyecto de destino `aws_demos.uvproj` en el directorio *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos`.
   - Para abrir el proyecto de demostración de Ethernet, elija el proyecto de destino `aws_demos_eth.uvproj` en el directorio *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth`.
4. Para asegurarse de que la configuración sea la correcta para instalar la placa, haga clic con el botón derecho en el proyecto `aws_demo` del IDE y, a continuación, elija Options (Opciones). (Consulte Solución de problemas para obtener más detalles).

5. En la pestaña Utilities (Utilidades), compruebe que Use Target Driver for Flash Programming (Utilizar el controlador de destino para la programación flash) esté seleccionado y que Nuvoton Nu-Link Debugger (Depurador de Nuvoton Nu-Link) esté definido como el controlador de destino.
6. En la pestaña Debug (Depurar), junto a Nuvoton Nu-Link Debugger (Depurador de Nuvoton Nu-Link), elija Settings (Configuración).
7. Compruebe que Chip Type (Tipo de chip) esté definido en M480.
8. En el panel de navegación Project (Proyecto) del IDE de Keil µVision, elija el proyecto `aws_demos`. En el menú Project (Proyecto), elija Build Target (Compilar destino).

Puede utilizar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Para ejecutar el proyecto de demostración de FreeRTOS

1. Conecte la placa Numaker-IoT-M487 al equipo (ordenador) host.
2. Vuelva a compilar el proyecto.
3. En el IDE de Keil µVision, en el menú Flash, elija Download (Descargar).
4. En el menú Debug (Depurar), elija Start/Stop Debug Session (Iniciar/detener sesión de depuración).
5. Cuando el depurador se detiene en el punto de ruptura en `main()`, abra el menú Run (Ejecutar) y, a continuación, elija Run (F5) (Ejecutar).

   Debería ver los mensajes de MQTT enviados por su dispositivo en el cliente de MQTT de la consola de AWS IoT.

Uso de CMake con FreeRTOS

También puede usar CMake para crear y ejecutar las aplicaciones de demostración de FreeRTOS o las aplicaciones que ha desarrollado con herramientas de depuración y editores de código de terceros.

Asegúrese de tener instalado el sistema de compilación CMake. Siga las instrucciones de Uso de CMake con FreeRTOS y, a continuación, siga los pasos de esta sección.

> **Note**
>
> Asegúrese de que la ruta a la ubicación del compilador (Keil) se encuentre en la variable del sistema Path, por ejemplo, `C:\Keil_v5\ARM\ARMCC\bin`.

También puede utilizar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Para generar archivos de compilación a partir de archivos de origen y ejecutar el proyecto de demostración

1. En la máquina host, abra el símbolo del sistema y vaya a la carpeta *freertos*.
2. Cree una carpeta para contener el archivo de compilación generado. Nos referiremos a esta carpeta como *BUILD_FOLDER*.
3. Genere los archivos de compilación para la demostración de Wi-Fi o Ethernet.

   - Para Wi-Fi:

     Desplácese hasta el directorio que contiene los archivos de código fuente del proyecto de demostración de FreeRTOS. A continuación, genere los archivos de compilación ejecutando el siguiente comando.

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -
B BUILD_FOLDER -G Ninja
```

- Para Ethernet:

  Desplácese hasta el directorio que contiene los archivos de código fuente del proyecto de demostración de FreeRTOS. A continuación, genere los archivos de compilación ejecutando el siguiente comando.

  ```
  cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -
  DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
  ```

4. Genere el binario para escribir en el M487 ejecutando el siguiente comando.

   ```
   cmake --build BUILD_FOLDER
   ```

   En este momento, el archivo binario `aws_demos.bin` debería encontrarse en la carpeta *BUILD_FOLDER*`/vendors/Nuvoton/boards/numaker_iot_m487_wifi`.

5. Para configurar la placa para el modo flash, asegúrese de que el conmutador MSG (n.º 4 de ISW1 en ICE) esté activado. Cuando conecte la placa, se asignará una ventana (y una unidad). (Consulte Solución de problemas).

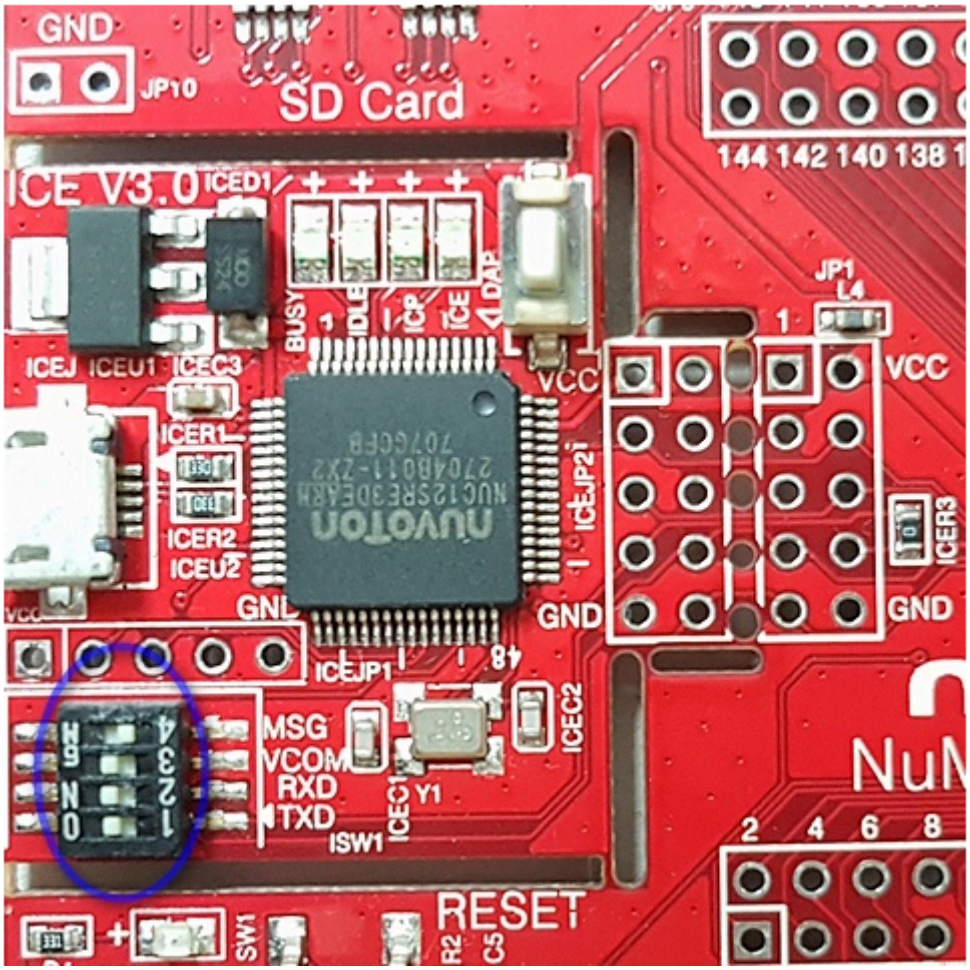
6. Abra un emulador de terminal para ver los mensajes a través de UART. Siga las instrucciones de Instalación de un emulador de terminal.

7. Ejecute el proyecto de demostración copiando el binario generado en el dispositivo.

   Si se ha suscrito al tema de MQTT con el cliente MQTT de AWS IoT, debe ver los mensajes MQTT enviados por su dispositivo en la consola de AWS IoT.

Solución de problemas

- Si Windows no reconoce el dispositivo `VCOM`, instale el controlador del puerto serie de Windows de NuMaker desde el enlace Nu-Link USB Driver v1.6.
- Si conecta su dispositivo a Keil MDK (el IDE) a través de Nu-Link, asegúrese de que el conmutador MSG (n.º 4 de ISW1 en ICE) esté desactivado, como se muestra.



Si tiene problemas para configurar su entorno de desarrollo o conectarse a la placa, póngase en contacto con Nuvoton.

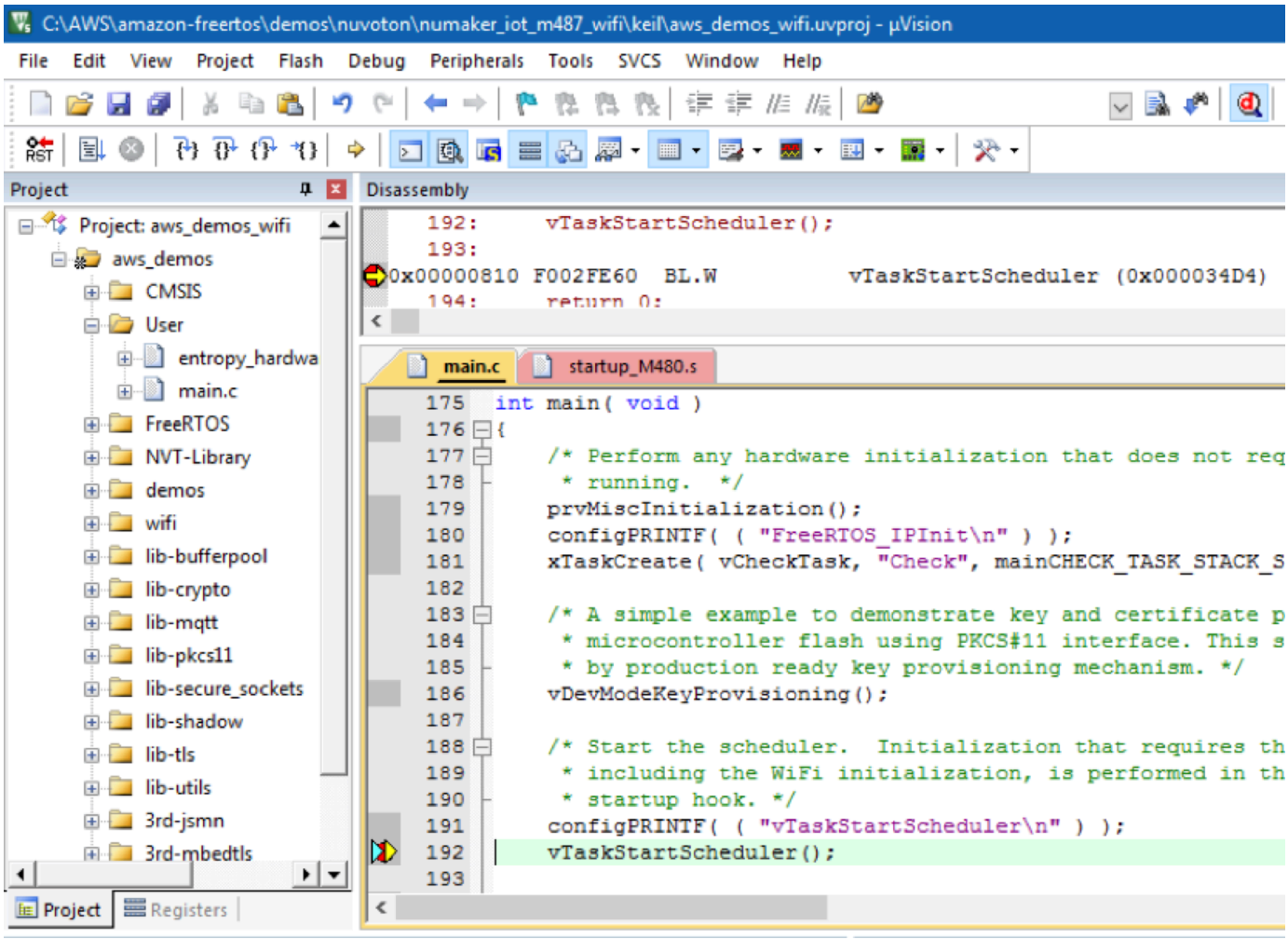
Depuración de proyectos de FreeRTOS en Keil µVision

Para iniciar una sesión de depuración en Keil µVision

1. Abra Keil µVision.
2. Siga los pasos para crear el proyecto de demostración de FreeRTOS en Creación y ejecución del proyecto de demostración de FreeRTOS.
3. En el menú Debug (Depurar), elija Start/Stop Debug Session (Iniciar/detener sesión de depuración).

   La ventana Call Stack+Locals (Pilas de llamadas + Variables locales aparece al iniciar una sesión de depuración. µVision instala la demostración en la placa, ejecuta la demostración y se detiene al inicio de la función `main()`.

4. Defina puntos de interrupción en el código fuente del proyecto y, a continuación, ejecute el código. El proyecto debería tener un aspecto similar al siguiente.



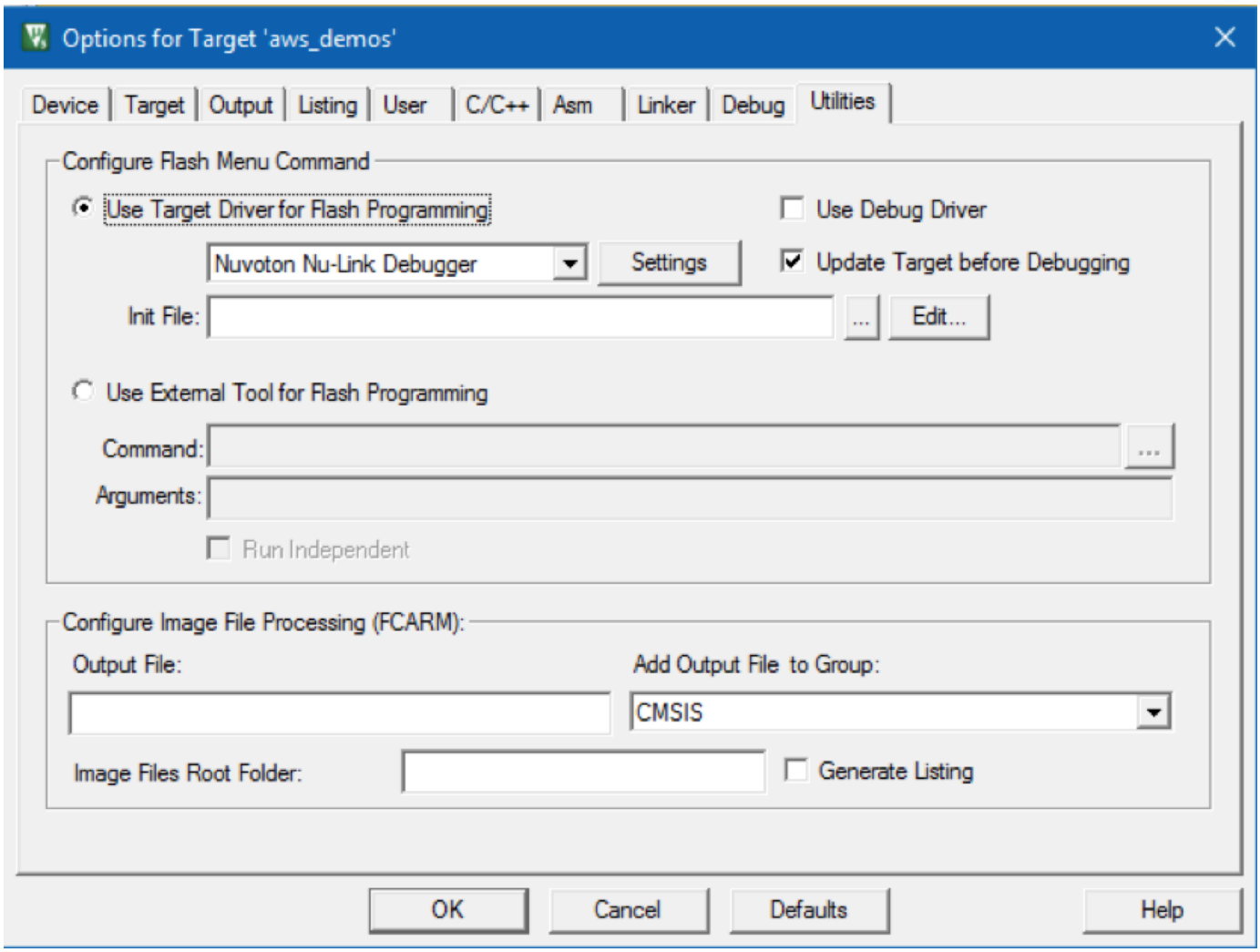
Solución de problemas de la configuración de depuración de µVision

Si tiene problemas al depurar una aplicación, compruebe que la configuración de depuración esté definida correctamente en Keil µVision.

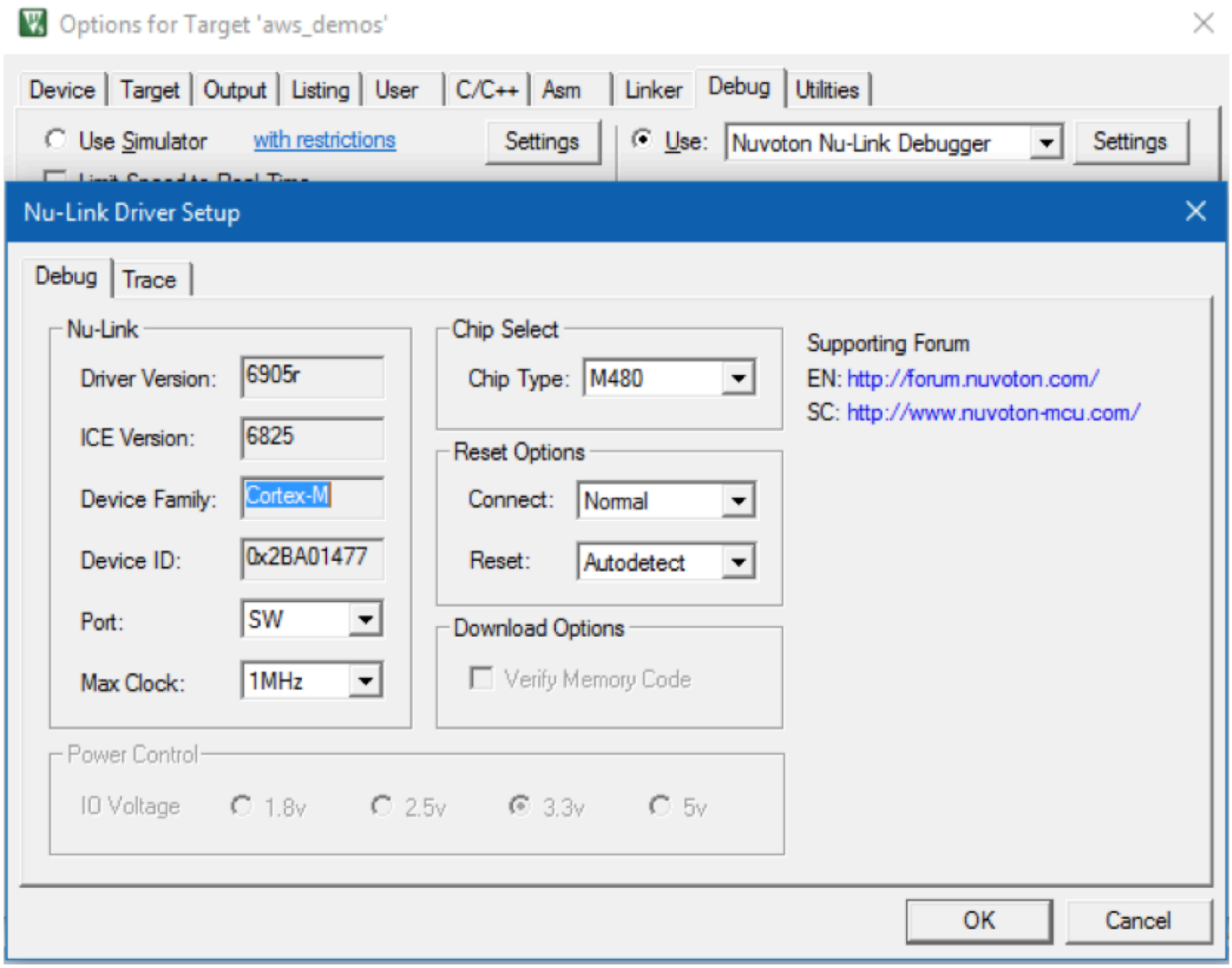
Para comprobar que la configuración de depuración de µVision sea la correcta

1. Abra Keil µVision.
2. Haga clic con el botón derecho en el proyecto `aws_demo` del IDE y, a continuación, elija Options (Opciones).
3. En la pestaña Utilities (Utilidades), compruebe que Use Target Driver for Flash Programming (Utilizar el controlador de destino para la programación flash) esté seleccionado y que Nuvoton

Nu-Link Debugger (Depurador de Nuvoton Nu-Link) esté definido como el controlador de destino.



4. En la pestaña Debug (Depurar), junto a Nuvoton Nu-Link Debugger (Depurador de Nuvoton Nu-Link), elija Settings (Configuración).



5. Compruebe que Chip Type (Tipo de chip) esté definido en M480.

Introducción al módulo de IoT LPC54018 de NXP

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción al módulo de IoT LPC54018 de NXP. Si no tiene un módulo de IoT NXP LPC54018, consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestro socio. Utilice un cable USB para conectar su módulo IoT LPC54018 de NXP a su equipo.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Conexión de su placa a un equipo host.
2. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
3. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
4. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.

Configuración del hardware de NXP

Para configurar LPC54018 de NXP

- Conecte su equipo al puerto USB de LPC54018 de NXP.

Para configurar el depurador JTAG

Necesita un depurador JTAG para lanzar y depurar el código que se ejecuta en la placa NXP LPC54018. FreeRTOS se probó con un módulo de IoT OM40006. Para obtener más información acerca de los depuradores compatibles, consulte el Manual del usuario para el módulo de IoT LPC54018 de NXP que está disponible en la página del producto OM40007 LPC54018 IoT Module .

1. Si utiliza un módulo de IoT OM40006, use un cable conversor para conectar el conector de 20 clavijas del depurador al conector de 10 clavijas del módulo de IoT de NXP.
2. Conecte el LPC54018 de NXP y el módulo de IoT OM40006 a los puertos USB en su equipo con cables de mini-USB a USB.

Configure el entorno de desarrollo.

FreeRTOS admite dos IDE para el módulo de IoT NXP LPC54018: IAR Embedded Workbench y MCUXpresso.

Antes de comenzar, instale uno de estos IDE.

Para instalar IAR Embedded Workbench for ARM

1. Busque IAR Embedded Workbench para ARM y descargue el software.

   > **Note**
   >
   > IAR Embedded Workbench for ARM requiere Microsoft Windows.

2. Ejecute el instalador y siga las instrucciones.
3. En el License Wizard (Asistente de licencia), elija Register with IAR Systems to get an evaluation license (Registrar en IAR Systems para obtener una licencia de evaluación).
4. Coloque el cargador de arranque en el dispositivo antes de intentar ejecutar cualquier demostración.

Para instalar MCUXpresso desde NXP

1. Descargue y ejecute el instalador MCUXpresso directamente desde NXP.

   > **Note**
   >
   > Las versiones 10.3.x y posteriores son compatibles.

2. Vaya a MCUXpresso SDK y elija Build your SDK (Cree su SDK).

   > **Note**
   >
   > Las versiones 2.5 y posteriores son compatibles.

3. Elija Select Development Board (Seleccionar placa de desarrollo).
4. En Select Development Board (Seleccionar placa de desarrollo), en Search by Name (Buscar por nombre), escriba **`LPC54018-IoT-Module`**.
5. En Boards (Placas), elija LPC54018-IoT-Module (Módulo de IoT LPC54018).

6. Compruebe los detalles del hardware y, a continuación, elija Build MCUXepresso SDK (Crear SDK de MCUXepresso).
7. El SDK para Windows mediante el IDE de MCUXpresso IDE ya se ha creado. Elija Download SDK. Si utiliza otro sistema operativo, en Host OS (SO host), seleccione su sistema operativo y, a continuación, elija Download SDK (Descargar SDK).
8. Inicie el IDE de MCUXpresso y elija la pestaña Installed SDKs (SDK instalados).
9. Arrastre y coloque el archivo de almacenamiento de SDK descargado en la ventana Installed SDKs (SDK instalados).

Si experimenta problemas durante la instalación, consulte NXP Support (Soporte de NXP) o NXP Developer Resources (Recursos para desarrolladores de NXP).

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje “¡Hola, mundo!” enviado varias veces al tema al que se ha suscrito.

Creación y ejecución del proyecto de demostración de FreeRTOS

Importación de la demostración de FreeRTOS en su IDE

Para importar el código de muestra de FreeRTOS al IDE IAR Embedded Workbench

1. Abra IAR Embedded Workbench y desde el menú File (Archivo) elija Open Workspace (Abrir espacio de trabajo).
2. En el cuadro de texto search-directory (buscar-directorio), escriba `projects/nxp/lpc54018iotmodule/iar/aws_demos` y elija aws_demos.eww.

3. En el menú Project (Proyecto) elija Rebuild All (Reconstruir todo).

Para importar el código de muestra de FreeRTOS al IDE de MCUXpresso

1. Abra MCUXpresso y en el menú File (Archivo), elija Open Projects From File System (Abrir proyectos del sistema de archivos).
2. En el cuadro de texto Directory (Directorio), escriba `projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos` y elija Finish (Finalizar)
3. En el menú Project (Proyecto) elija Build All (Compilar todo).

Ejecución del proyecto de demostración de FreeRTOS

Para ejecutar el proyecto de demostración de FreeRTOS con el IDE IAR Embedded Workbench

1. En el IDE, en el menú Project (Proyecto) elija Make (Hacer).
2. Desde el menú Project (Proyecto), elija Download y Debug (Descargar y depurar).
3. En el menú Depurar, elija Iniciar depuración.
4. Cuando el depurador se detenga en el punto de ruptura en `main`, desde el menú Debug (Depurar), elija Go (Ir).

> **Note**
>
> Si se abre un cuadro de diálogo J-Link Device Selection (Selección del dispositivo J-Link), haga clic en OK para continuar. En el cuadro de diálogo Target Device Settings (Configuración del dispositivo de destino), elija Unspecified (Sin especificar), elija Cortex-M4 y, a continuación, haga clic en OK. Solo tiene que hacerlo una vez.

Para ejecutar el proyecto de demostración de FreeRTOS con el IDE MCUxpresso

1. En su IDE, en el menú Proyecto elija Compilar.
2. Si es la primera vez que realiza la depuración, elija el proyecto `aws_demos` y desde la barra de herramientas de depuración, elija el botón de depuración azul.
3. Se muestra cualquier sondeo de depuración detectado. Elija la sonda que desea utilizar y, a continuación, haga clic en OK para iniciar la depuración.

> **Note**
>
> Cuando el depurador se detenga en el punto de interrupción `main()`, pulse el botón de reinicio de depuración en
>
> 
>
> una vez para restablecer la sesión de depuración. (Esto es necesario debido a un error con el depurador MCUXpresso para NXP54018-IoT-Module).

4. Cuando el depurador se detenga en el punto de ruptura en `main()`, desde el menú Debug (Depurar), elija Go (Ir).

Solución de problemas

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción a Renesas Starter Kit+ for RX65N-2MB

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción a Renesas Starter Kit+ para RX65N-2MB. Si no tiene Renesas RSK+ para RX65N-2MB, consulte el Catálogo de dispositivos de socios de AWS y adquiera uno de nuestros socios.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *`freertos`*.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Conexión de su placa a un equipo host.
2. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
3. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
4. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.

Configuración de hardware Renesas

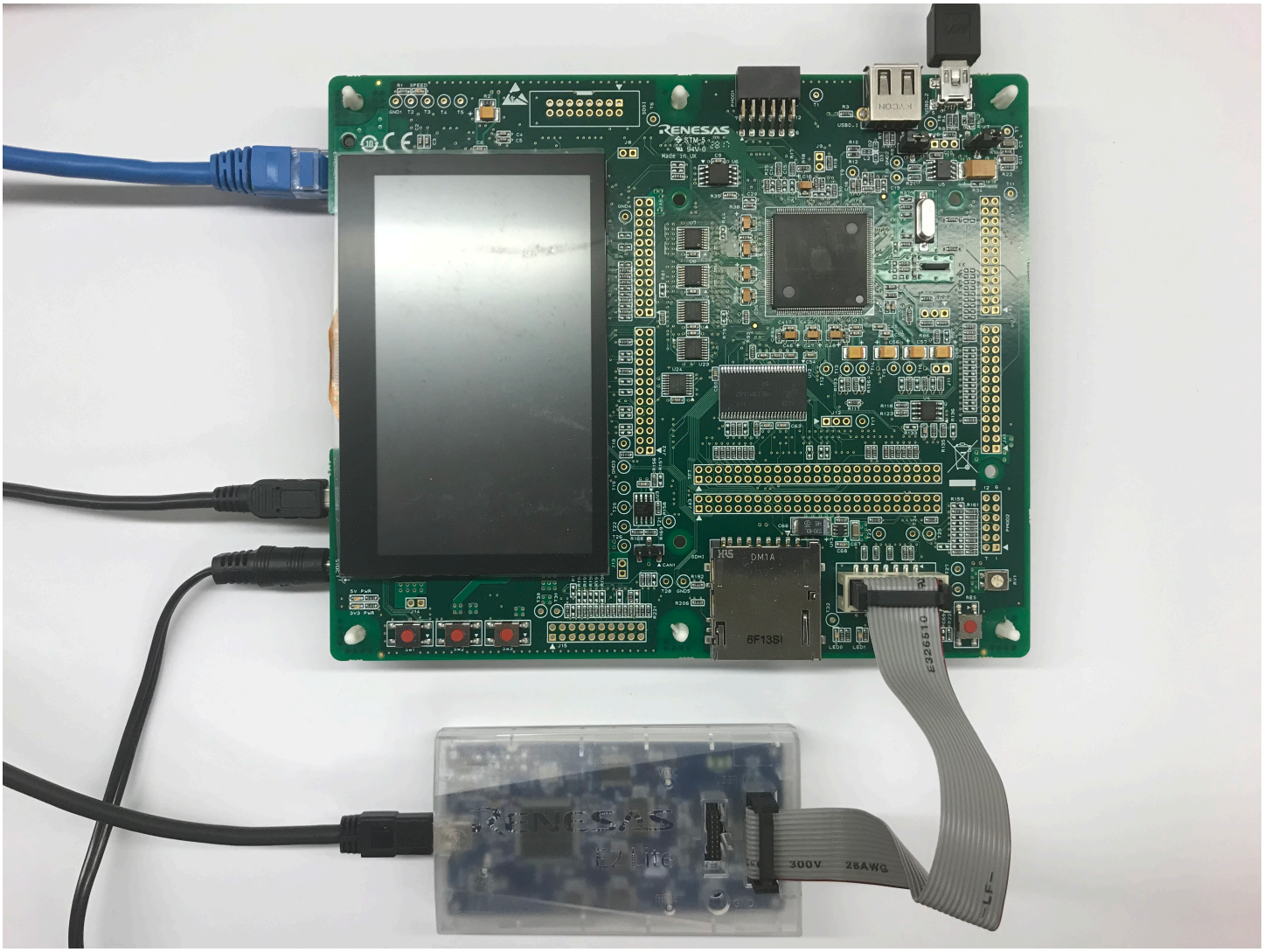
Para configurar el RSK+ for RX65N-2MB

1. Conecte el adaptador de alimentación +5V positivo al conector PWR en la RSK+for RX65N-2MB.
2. Conecte el equipo al puerto USB2.0 FS en la RSK+ for USB2-RX65N.
3. Conecte el equipo al puerto USB a serial en la RSK+ for RX65N-2MB.
4. Conecte un router o un puerto Ethernet conectado a Internet al puerto Ethernet en la RSK+ for RX65N-2MB.

Para configurar el módulo del depurador E2 Lite

1. Utilice el cable de 14 pines para conectar el módulo del depurador E2 Lite al puerto ‘E1/E2 Lite’ en la RSK+for RX65N-2MB.
2. Utilice un cable USB para conectar el módulo del depurador E2 Lite a su máquina host. Cuando el depurador E2 Lite está conectado a la placa y al equipo, un LED verde "ACT" parpadea en el depurador.
3. Después de que el depurador esté conectado a su equipo host y a RSK+for RX65N-2MB, comienza la instalación de los controladores del depurador E2 Lite.

   Tenga en cuenta que se requieren privilegios de administrador para instalar los controladores.



Configure el entorno de desarrollo.

Para establecer las configuraciones de FreeRTOS para RSK+ para RX65N-2MB, utilice el IDE de Renesas e^{2}studio y el compilador CC-RX.

> **Note**
>
> El compilador Renesas e^{2}studio IDE y CC-RX solo es compatible con los sistemas operativos Windows 7, 8 y 10.

Para descargar e instalar e^{2}studio

1. Vaya a la página de descarga del Instalador de Renesas e^{2}studio y descargue el instalador sin conexión.
2. Le redirigirá a la página de inicio de sesión de Renesas.

   Si tiene una cuenta con Renesas, introduzca sus credenciales de inicio de sesión y elija Iniciar sesión.

   Si no tiene una cuenta, elija Register now (Regístrese ahora) y siga los primeros pasos de registro. Debería recibir un correo electrónico con un enlace para activar su cuenta de Renesas. Haga clic en este enlace para completar tu registro en Renesas y, a continuación, inicie sesión en Renesas.
3. Después de iniciar sesión, descargue el instalador de e^{2}studio en su equipo.
4. Abra el instalador y siga los pasos de instalación.

Para obtener más información, consulte e^{2}studio en el sitio web de Renesas.

Para descargar e instalar el paquete RX Family C/C++ Compiler Package

1. Vaya a la página de descarga de RX Family C/C++ Compiler Package y descargue el paquete V3.00.00.
2. Abra el archivo ejecutable e instale el compilador.

Para obtener más información, consulte C/C++ Compiler Package for RX Family en el sitio web de Renesas.

> **Note**
>
> El compilador solo está disponible para la versión gratuita de evaluación y tiene una validez de 60 días. En el sexagésimo primer día, debe obtener una clave de licencia. Para obtener más información, consulte Evaluation Software Tools.

Creación y ejecución de muestras de FreeRTOS

Ahora que ha configurado el proyecto de demostración, todo está listo para compilar y ejecutar el proyecto en la placa.

Creación de la demostración de FreeRTOS en e^{2}studio

Para importar y compilar la demostración en e^{2}studio

1. Inicie e^{2}studio en el menú Inicio.
2. En la ventana Select a directory as a workspace (Seleccionar un directorio como espacio de trabajo), examine la carpeta en la que desea trabajar y elija Launch (Lanzar).
3. La primera vez que abra e^{2}studio, se abre la ventana Toolchain Registry (Registro de la cadena de herramientas). Elija Renesas Toolchains (Cadenas de herramientas de Renesas) y confirme que se selecciona **`CC-RX v3.00.00`**. Elija Register (Registrar) y, después, OK.
4. Si abre e^{2}studio por primera vez, aparece la ventana Code Generator Registration (Registro del generador de código). Seleccione OK (Aceptar).
5. Aparece la ventana Code Generator COM component register (Registrador de componente COM de generador de código). En Reiniciar e^{2}studio para utiliza el generador de código, elija Aceptar.
6. Se muestra la ventana Reiniciar e ^{2}studio. Seleccione OK (Aceptar).
7. e^{2}studio se reinicia. En la ventana Select a directory as a workspace (Seleccionar un directorio como espacio de trabajo), elija Launch (Iniciar).
8. En la pantalla de bienvenida de e^{2}studio, seleccione el icono de flecha Ir a e^{2}studio workbench.
9. Haga clic con el botón derecho del ratón en la ventana Project Explorer (Explorador de proyectos) y elija Import (Importar).
10. En el asistente de importación, elija General, elija Existing Projects into Workspace (Proyectos existentes a Workspace) y, a continuación, elija Next (Siguiente).
11. Elija Browse (Examinar), localice el directorio `projects/renesas/rx65n-rsk/e2studio/aws_demos` y, a continuación, elija Finish (Finalizar).
12. En el menú Project (Proyecto) Project (Proyecto), Build All (Compilar todo).

    La consola de compilación emite un mensaje de advertencia de que License Manager no está instalado. Puede hacer caso omiso de este mensaje a menos que tenga una clave de licencia para el compilador CC-RX. Para instalar License Manager, consulte la página de descarga de License Manager.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje “¡Hola, mundo!” enviado varias veces al tema al que se ha suscrito.

Ejecución del proyecto FreeRTOS

Para ejecutar el proyecto en e^{2}studio

1. Confirme que ha conectado el módulo del depurador E2 Lite a su RSK+for RX65N-2MB
2. En el menú superior, elija Run (Ejecutar), Debug Configurations (Configuraciones de depuración).
3. Amplíe Renesas GDB Hardware Debugging (Depuración de hardware GDB de Renesas) y elija aws_demos HardwareDebug.
4. Elija la pestaña Debugger (Depurador) y, a continuación, elija la pestaña Connection Settings (Configuración de conexión). Confirme que su configuración de conexión sea correcta.
5. Elija Debug (Depurar) para descargar el código en la placa y comenzar la depuración.

   Es posible que se solicite `e2-server-gdb.exe` mediante una advertencia de firewall. Compruebe Private networks, such as my home or work network (Redes privadas, como mi red doméstica o de trabajo) y, a continuación, seleccione Allow access (Permitir acceso).
6. e^{2}studio podría pedir cambia a Renesas Debug Perspective (Perspectivas de depuración de Renesas). Seleccione Yes.

   El LED green de "ACT" en el depurador E2 Lite se ilumina.

7. Una vez que el código se descarga en la placa, seleccione Resume (Reanudar) para ejecutar el código hasta la primera línea de la función principal. Seleccione Resume (Reanudar) de nuevo para ejecutar el resto del código.

Para obtener la información más reciente acerca de proyectos publicado por Renesas, consulte la bifurcación `renesas-rx` del repositorio `amazon-freertos` en GitHub.

Solución de problemas

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción al STMicroelectronics STM32L4 Discovery Kit IoT Node

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción a STMicroelectronics STM32L4 Discovery Kit IoT Node. Si todavía no tiene el nodo de IoT STMicroelectronics STM32L4 Discovery Kit, consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestro socio.

Asegúrese de tener instalada la última versión de firmware de Wi-Fi. Para descargar el firmware de Wi-Fi más reciente, consulte STM32L4 Discovery kit IoT node, low-power wireless, Bluetooth Low Energy, NFC, SubGHz, Wi-Fi. Bajo Binary Resources (Recursos binarios), elija Inventek ISM 43362 Wi-Fi module firmware update (read the readme file for instructions) (Actualización de firmware del módulo wifi ISM 43362 de Inventek [lea el archivo Léame para obtener instrucciones]).

Antes de comenzar, debe configurar AWS IoT, la descarga de FreeRTOS y Wi-Fi para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *`freertos`*.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
2. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
3. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.

Configure el entorno de desarrollo.

Instalación de System Workbench para STM32

1. Vaya a OpenSTM32.org.
2. Regístrese en la página web OpenSTM32. Debe para iniciar sesión para descargar System Workbench.
3. Vaya al instalador de System Workbench para STM32 para descargar e instalar System Worbench.

Si tiene problemas durante la instalación, consulte las preguntas frecuentes en el sitio web de System Workbench.

Creación y ejecución del proyecto de demostración de FreeRTOS

Importación de la demostración de FreeRTOS a STM32 System Workbench

1. Abra STM32 System Workbench y escriba un nombre para un espacio de trabajo nuevo.
2. En el menú File, elija Import. Expanda General, elija Existing Projects into Workspace (Proyectos existentes a Workspace) y, a continuación, elija Next (Siguiente).
3. En Select Root Directory (Seleccionar directorio raíz), escriba `projects/st/stm32l475_discovery/ac6/aws_demos`.
4. El proyecto `aws_demos` debe seleccionarse de forma predeterminada.
5. Elija Finish (Finalizar) para importar el proyecto a STM32 System Workbench.
6. En el menú Project (Proyecto) elija Build All (Compilar todo). Confirme que el proyecto se compila sin ningún error.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje “¡Hola, mundo!” enviado varias veces al tema al que se ha suscrito.

Ejecución del proyecto de demostración de FreeRTOS

1. Utilice un cable USB para conectar su STMicroelectronics STM32L4 Discovery Kit IoT Node a su equipo. (Consulte la documentación del fabricante que viene con la placa para saber cuál es el puerto USB correcto que debe utilizar).
2. Desde Project Explorer (Explorador de proyectos), haga clic con el botón derecho del ratón en `aws_demos`, elija Debug As (Depurar como) y, a continuación, elija Ac6 STM32 C/C++ Application (Aplicación Ac6 STM32 C/C++).

   Si se produce un error de depuración al lanzar por primera vez una sesión de depuración, siga estos pasos:

   1. En System Workbench STM32, en el menú Run (Ejecutar), elija Debug Configurations (Configuraciones de depuración).
   2. Elija aws_demos Debug (Depurar demos_aws). (Es posible que tenga que ampliar Ac6 STM32 Debugging (Depuración de Ac6 STM32).)
   3. Elija la pestaña Debugger (Depurador).
   4. En Configuration Script (Script de configuración), elija Show Generator Options (Mostrar opciones del generador).

5. En Mode Setup (Configuración de modo), establezca Reset Mode (Modo de restablecimiento) en Software System Reset (Restablecimiento de sistema de software). Elija Apply (Aplicar) y, a continuación, Debug (Depurar).

3. Cuando el depurador se detenga en el punto de ruptura en `main()`, desde el menú Run (Ejecutar), elija Resume (Reanudar).

Uso de CMake con FreeRTOS

Si prefiere no utilizar un IDE para el desarrollo de FreeRTOS, también puede usar CMake para crear y ejecutar las aplicaciones de demostración o las aplicaciones que ha desarrollado con herramientas de depuración y editores de código de terceros.

En primer lugar, cree una carpeta para incluir los archivos de compilación generados (*carpeta-compilación*).

Utilice el siguiente comando para generar archivos de compilación:

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-folder
```

Si `arm-none-eabi-gcc` no se encuentra en la ruta del shell, también debe configurar la variable `AFR_TOOLCHAIN_PATH` de CMake. Por ejemplo:

```
-D AFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

Para obtener más información sobre cómo usar CMake con FreeRTOS, consulte Uso de CMake con FreeRTOS.

Solución de problemas

Si ve lo siguiente en la salida UART de la aplicación de demostración, debe actualizar el firmware del módulo wifi:

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

Para descargar el firmware de Wi-Fi más reciente, consulte STM32L4 Discovery kit IoT node, low-power wireless, Bluetooth Low Energy, NFC, SubGHz, Wi-Fi. En Binary Resources (Recursos

binarios), elija el enlace de descarga para Inventek ISM 43362 Wi-Fi module firmware update (Actualización de firmware del módulo wifi 43362 ISM de Inventek).

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción a CC3220SF-LAUNCHXL de Texas Instruments

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción a CC3220SF-LAUNCHXL de Texas Instruments. Si todavía no tiene el kit de desarrollo CC3220SF-LAUNCHXL de Texas Instruments (TI), consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestro socio.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *`freertos`*.

Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
2. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
3. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.

Configure el entorno de desarrollo.

Siga los pasos que se indican a continuación para configurar el entorno de desarrollo para comenzar a usar FreeRTOS.

Tenga en cuenta que FreeRTOS admite dos IDE para el kit de desarrollo TI CC3220SF-LAUNCHXL: Code Composer Studio e IAR Embedded Workbench, versión 8.32. Puede utilizar cualquier IDE para comenzar.

Instale Code Composer Studio

1. Vaya a TI Code Composer Studio.
2. Descargue el instalador sin conexión para la plataforma de su equipo host (Windows, macOS o Linux 64 bits).
3. Descomprima y ejecute el instalador sin conexión. Siga las instrucciones.
4. Para instalar las familias de productos, elija las MCU inalámbricas SimpleLink Wi-Fi CC32xx.
5. En la página siguiente, acepte la configuración predeterminada para sondas de depuración y, a continuación, elija Finalizar.

Si experimenta problemas durante la instalación de Code Composer Studio, consulte TI Development Tools Support, Code Composer Studio FAQs y Troubleshooting CCS.

Instalar IAR Embedded Workbench

1. Descargue y ejecute el instalador de Windows para la versión 8.32 de IAR Embedded Workbench para ARM. En Debug probe drivers (Depurar controladores de sondeo), asegúrese de haber seleccionado TI XDS (XDS de TI).
2. Complete la instalación y lance el programa. En la página License Wizard (Asistente de licencias), elija Register with IAR Systems to get an evaluation license (Regístrese en IAR Systems para obtener una licencia de evaluación) o utilice su propia licencia de IAR.

Instale el CC3220 SDK SimpleLink

Instale el SDK del SimpleLink CC3220. El SDK CC3220 de SimpleLink Wi-Fi contiene los controladores para la MCU programable CC3220SF, más de 40 aplicaciones de muestra y la documentación necesaria para utilizar las muestras.

Instale Uniflash

Instale Uniflash CCS Uniflash es una herramienta independiente que se utiliza para programar memoria flash en chip en MCU de TI. Uniflash tiene una GUI, línea de comandos e interfaz de scripting.

Instale el Service Pack más reciente

1. En su CC3220SF-LAUNCHXL de TI, coloque el puente SOP en el centro del conjunto de pines (posición = 1) y restablezca la placa.
2. Comience Uniflash. Si su placa LaunchPad CC3220SF aparece en Dispositivos detectados, seleccione Iniciar. Si no se detecta la placa, elija CC3220SF-LAUNCHXL en la lista de placas en New Configuration (Nueva configuración), a continuación, elija Start Image Creator (Iniciar creador de imagen).
3. Elija New Project (Nuevo proyecto).
4. En la página Start new project (Comenzar proyecto nuevo), escriba un nombre para el proyecto. En Device Type (Tipo de dispositivo), seleccione CC3220SF. En Device Mode (Modo de dispositivo), elija Develop (Desarrollar) y, a continuación, elija Create Project (Crear proyecto).
5. En el lado derecho de la ventana de la aplicación Uniflash, elija Connect (Conectar).
6. En la columna izquierda, seleccione Advanced (Avanzados), Files (Archivos) y, a continuación, Service Pack.
7. Selecciona Buscar y, a continuación, navega hasta el lugar donde instalaste el SDK del CC3220SF. SimpleLink El Service Pack se encuentra en `ti/simplelink_cc32xx_sdk_`*`VERSION`*`/tools/cc32xx_tools/servicepack-cc3x20/sp_`*`VERSION`*`.bin`.
8. Elija el botón Burn (Grabar) () y, a continuación, elija Program Image (Create & Program) (Programar imagen [Crear y programar]) para instalar el Service Pack. Recuerde cambiar el puente SOP a la posición 0 y restablecer la placa.

Configurar el aprovisionamiento de wifi

Para configurar los ajustes de wifi para la placa, realice una de las siguientes operaciones:

- Configure la aplicación de demostración de FreeRTOS que se describe en Configuración de las demostraciones de FreeRTOS.
- Úselo SmartConfigdesde Texas Instruments.

Creación y ejecución del proyecto de demostración de FreeRTOS

Creación y ejecución del proyecto de demostración de FreeRTOS en TI Code Composer

Par importar la demostración de FreeRTOS en TI Code Composer

1. Abra TI Code Composer y haga clic en OK para aceptar el nombre del espacio de trabajo predeterminado.
2. En la página Getting started (Introducción), elija Import Project (Importar proyecto).
3. En Select search-directory (Seleccionar directorio de búsqueda), escriba `projects/ti/cc3220_launchpad/ccs/aws_demos`. El proyecto `aws_demos` debe seleccionarse de forma predeterminada. Para importar el proyecto a TI Code Composer, elija Finish (Finalizar).
4. En Project Explorer (Explorador de proyectos), haga doble clic en aws_demos para activar el proyecto.
5. Desde Project (Proyecto), elija Build Project (Crear proyecto) para asegurarse de que el proyecto se compila sin errores o advertencias.

Para ejecutar la demostración de FreeRTOS en TI Code Composer

1. Asegúrese de que el puente Sense On Power (SOP) de su CC3220SF-LAUNCHXL de Texas Instruments está en la posición 0. Para obtener más información, consulte la Guía del usuario del procesador de SimpleLink red Wi-Fi CC3x20 y CC3x3x.
2. Utilice un cable USB para conectar su CC3220SF-LAUNCHXL de Texas Instruments a su equipo.
3. En el explorador de proyectos, asegúrese de que `CC3220SF.ccxml` se ha seleccionado como configuración de destino activa. Para que activarla, haga clic con el botón derecho en el archivo y seleccione Set as active target configuration (Establecer como configuración de destino activa).
4. En TI Code Composer, desde Run (Ejecutar), elija Debug (Depurar).
5. Cuando el depurador se detiene en el punto de ruptura en `main()`, diríjase al menú Run (Ejecutar) y, a continuación, elija Resume (Reanudar).

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje "¡Hola, mundo!" enviado varias veces al tema al que se ha suscrito.

Creación y ejecución del proyecto de demostración de FreeRTOS en IAR Embedded Workbench

Para importar la demostración de FreeRTOS en IAR Embedded Workbench

1. Abra IAR Embedded Workbench, elija File (Archivo) y, a continuación elija Open Workspace (Abrir Workspace).
2. Vaya a `projects/ti/cc3220_launchpad/iar/aws_demos`, elija aws_demos.eww y, a continuación, haga clic en OK (Aceptar).
3. Haga clic con el botón derecho en el nombre del proyecto (`aws_demos`) y, a continuación, elija Make (Crear).

Para ejecutar la demostración de FreeRTOS en IAR Embedded Workbench

1. Asegúrese de que el puente Sense On Power (SOP) de su CC3220SF-LAUNCHXL de Texas Instruments está en la posición 0. Para obtener más información, consulte la Guía del usuario del procesador de SimpleLink red Wi-Fi CC3x20, CC3x3x.
2. Utilice un cable USB para conectar su CC3220SF-LAUNCHXL de Texas Instruments a su equipo.
3. Vuelva a compilar el proyecto.

   Para volver a crear el proyecto, en el menú Project (Proyecto), elija Make (Crear).
4. Desde el menú Project (Proyecto), elija Download y Debug (Descargar y depurar). Si aparece el mensaje «Advertencia: no se pudo inicializar», puede omitir el mensaje «Advertencia: no se pudo EnergyTrace inicializar». Para obtener más información al respecto EnergyTrace, consulte Tecnología MSP. EnergyTrace

5. Cuando el depurador se detenga en el punto de ruptura en `main()`, diríjase al menú Debug (Depurar) y, a continuación, elija Go (Ir).

Uso de CMake con FreeRTOS

Si prefiere no utilizar un IDE para el desarrollo de FreeRTOS, también puede usar CMake para crear y ejecutar las aplicaciones de demostración o las aplicaciones que ha desarrollado con herramientas de depuración y editores de código de terceros.

Para crear la demostración de FreeRTOS con CMake

1. Cree una carpeta para contener los archivos de compilación generados (*carpeta-compilación*).
2. Asegúrese de que la ruta de búsqueda (variable de entorno $PATH) contiene la carpeta en la que se encuentra el binario del compilador de CGT de TI (por ejemplo, `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`).

   Si está utilizando el compilador ARM de TI con su placa de TI, utilice el siguiente comando para generar los archivos de compilación del código de origen:

   ```
   cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S freertos -B build-folder
   ```

Para obtener más información, consulte Uso de CMake con FreeRTOS.

Solución de problemas

Si no ve mensajes en el cliente de MQTT de la consola de AWS IoT, es posible que tenga que configurar las opciones de depuración para la placa.

Para configurar los ajustes de depuración de placas de TI

1. En Code Composer, en Project Explorer (Explorador de proyectos), elija aws_demos.
2. En el menú Run (Ejecutar), elija Debug Configurations (Configuraciones de depuración).
3. En el panel de navegación, elija aws_demos.
4. En la pestaña Target (Destino), bajo Connection Options (Opciones de conexión), elija Reset the target on a connect (Restablecer el destino en una conexión).
5. Seleccione Apply y, a continuación, seleccione Close.

Si estos pasos no funcionan, mire la salida del programa en el terminal de la serie. Debería ver un texto que indica el origen del problema.

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción al simulador de dispositivos de Windows

Este tutorial ofrece instrucciones sobre cómo empezar a utilizar el simulador de dispositivos de Windows de FreeRTOS.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *`freertos`*.

FreeRTOS se publica como un archivo zip que contiene las bibliotecas de FreeRTOS y aplicaciones de muestra para la plataforma que especifique. Para ejecutar las muestras en un equipo Windows, descargue las bibliotecas y las muestras transferidas para su ejecución en Windows. Este conjunto de archivos se denomina simulador de Windows de FreeRTOS.

> **Note**
>
> Este tutorial no se puede ejecutar correctamente en instancias de Windows de Amazon EC2.

Configure el entorno de desarrollo.

1. Instale la versión más reciente de Npcap. Seleccione el “Modo compatible con la API WinPcap” durante la instalación.
2. Instale Microsoft Visual Studio.

   Se sabe que las versiones 2017 y 2019 de Visual Studio funcionan. Todas las ediciones de estas versiones de Visual Studio son compatibles (Community, Professional o Enterprise).

   Además del IDE, instale el componente Desarrollo de escritorio con C++.

   Instale el SDK más reciente de Windows 10. Puede elegirlo en la sección Opcional del componente Desarrollo de escritorio con C++.
3. Asegúrese de que tiene un activo conexión cableados.

4. (Opional) Si desea utilizar el sistema de creación basado en CMake para crear sus proyectos de FreeRTOS, instale la versión más reciente de CMake. Se necesita la versión 3.13 o posterior de FreeRTOS para CMake.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Cuando el proyecto de demostración se ejecute correctamente en su dispositivo, verá el mensaje “¡Hola, mundo!” enviado varias veces al tema al que se ha suscrito.

Creación y ejecución del proyecto de demostración de FreeRTOS

Puede utilizar Visual Studio o CMake para crear proyectos de FreeRTOS.

Creación y ejecución de proyectos de demostración de FreeRTOS con el IDE de Visual Studio

1. Cargue el proyecto en Visual Studio.

   En Visual Studio, desde el menú Archivo, elija Abrir. Elija File/Solution (Archivo/Solución), vaya al archivo `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` y, a continuación, elija Open (Abrir).
2. Cambie el destino del proyecto de demostración.

   El proyecto de demostración proporcionado depende del SDK de Windows, pero no tiene una versión de SDK de Windows especificada. De forma predeterminada, el IDE podría intentar compilar la demostración con una versión del SDK que no se encuentra en su equipo. Para establecer la versión del SDK de Windows, haga clic con el botón derecho del ratón en `aws_demos` y, a continuación, elija Redestinar proyectos. Se abrirá la ventana Revisar acciones

de solución. Elija una versión del SDK de Windows que esté en su equipo (el valor inicial en el menú desplegable es suficiente) y, a continuación, elija Aceptar.

3. Compile y ejecute el proyecto.

   Desde el menú Build (Crear), elija Build Solution (Crear solución) y asegúrese de que la solución se crea sin errores ni advertencias. Elija Depurar, Iniciar depuración para ejecutar el proyecto. En la primera ejecución, deberá seleccionar una interfaz de red.

Creación y ejecución del proyecto de demostración de FreeRTOS con CMake

Le recomendamos que utilice la GUI de CMake en lugar de la herramienta de línea de comandos de CMake para compilar el proyecto de demostración para el simulador de Windows.

Después de instalar CMake, abra la GUI de CMake. En Windows, encontrará esto en el menú de Inicio en CMake, CMake (cmake-gui).

1. Configure el directorio de códigos fuente de FreeRTOS.

   En la GUI, establezca el directorio de códigos fuente de FreeRTOS (*`freertos`*) en Dónde está el código fuente.

   Establezca *`freertos`*`/build` para Where to build the binaries (Dónde compilar los binarios).

2. Configure el proyecto de CMake.

   En la GUI de CMake, elija Add Entry (Añadir entrada) y en la ventana Add Cache Entry (Añadir entrada de caché), defina los siguientes valores:

   Nombre

   AFR_BOARD

   Tipo

   STRING

   Valor

   pc.windows

   Descripción

   (Opcional)

3. Elija Configure. Si CMake le pide que cree el directorio de compilación, elija Yes (Sí) y, a continuación, seleccione un generador en Specify the generator for this project (Especificar el generador para este proyecto). Le recomendamos que utilice Visual Studio como generador, pero Ninja también es compatible. (Tenga en cuenta que al usar Visual Studio 2019, la plataforma debe establecerse en Win32 en lugar de en su configuración predeterminada). Deje el resto de opciones del generador sin cambios y elija Finalizar.
4. Genere y abra el proyecto de CMake.

   Una vez que haya configurado el proyecto, la GUI de CMake muestra todas las opciones disponibles para el proyecto generado. A efectos de este tutorial, puede dejar las opciones con sus valores predeterminados.

   Elija Generate (Generar) para crear una solución de Visual Studio y, a continuación, elija Open Project (Abrir proyecto) para abrir el proyecto en Visual Studio.

   En Visual Studio, haga clic con el botón derecho del ratón en el proyecto `aws_demos` y seleccione Establecer como proyecto de inicio. Esto le permite compilar y ejecutar el proyecto. En la primera ejecución, deberá seleccionar una interfaz de red.

Para obtener más información sobre cómo usar CMake con FreeRTOS, consulte Uso de CMake con FreeRTOS.

Configurar su interfaz de red

En la primera ejecución del proyecto de demostración, debe seleccionar la interfaz de red que va a utilizar. El programa hace un recuento de sus interfaces de red. Localice el número para la interfaz de Ethernet cableada. La salida debe tener el siguiente aspecto:

```
0 0 [None] FreeRTOS_IPInit
1 0 [None] vTaskStartScheduler
1. rpcap://\Device\NPF_{AD01B877-A0C1-4F33-8256-EE1F4480B70D}
(Network adapter 'Intel(R) Ethernet Connection (4) I219-LM' on local host)

2. rpcap://\Device\NPF_{337F7AF9-2520-4667-8EFF-2B575A98B580}
(Network adapter 'Microsoft' on local host)

The interface that will be opened is set by "configNETWORK_INTERFACE_TO_USE", which
should be defined in FreeRTOSConfig.h

ERROR:  configNETWORK_INTERFACE_TO_USE is set to 0, which is an invalid value.
```

```
Please set configNETWORK_INTERFACE_TO_USE to one of the interface numbers listed above,
then re-compile and re-start the application.  Only Ethernet (as opposed to Wi-Fi)
interfaces are supported.
```

Una vez que haya identificado el número de su interfaz de Ethernet cableada, cierre la ventana de la aplicación. En el ejemplo anterior, el número que se va a utilizar es 1.

Abra `FreeRTOSConfig.h` y establezca `configNETWORK_INTERFACE_TO_USE` en el número que corresponda a su interfaz de red cableada.

> **Important**
>
> Solo se admiten interfaces Ethernet. La conexión wifi no es compatible.

Solución de problemas

Solución de problemas de problemas comunes en Windows

Es posible que encuentre el siguiente error al intentar compilar el proyecto de demostración con Visual Studio:

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual
 Studio solution.
```

El proyecto debe realizarse con la versión de SDK de Windows presente en su equipo.

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Introducción al Xilinx Avnet MicroZed Industrial IoT Kit

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Este tutorial ofrece instrucciones para la introducción a Xilinx Avnet MicroZed Industrial IoT Kit Si no tiene el kit de Xilinx Avnet MicroZed Industrial, consulte el Catálogo de dispositivos de socios de AWS para adquirir uno de nuestro socio.

Antes de comenzar, debe configurar AWS IoT y la descarga de FreeRTOS para conectar el dispositivo a la nube de AWS. Para obtener instrucciones, consulte Primeros pasos. En este tutorial, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.

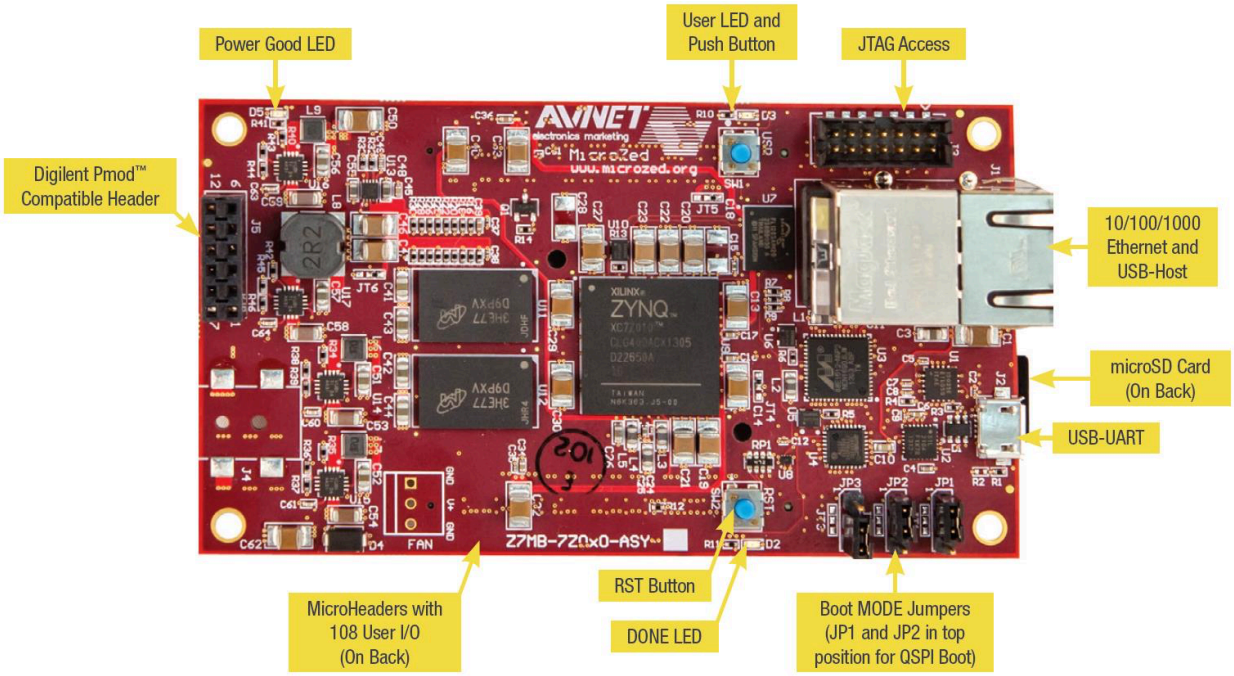
Información general

Este tutorial contiene instrucciones para los siguientes pasos de introducción:

1. Conexión de su placa a un equipo host.
2. Instalación de software en el equipo host para desarrollar y depurar las aplicaciones integradas de la placa de su microcontrolador.
3. Compilación cruzada de una aplicación de demostración de FreeRTOS en una imagen binaria.
4. Carga de la imagen binaria de la aplicación en su placa y, a continuación, ejecución de la aplicación.

Configuración de hardware MicroZed

El siguiente diagrama puede resultar útil al configurar el hardware MicroZed:



Para configurar la placa MicroZed

1. Conecte el equipo al puerto USB-UART en su placa MicroZed.
2. Conecte el equipo al puerto de acceso JTAG en su placa MicroZed.
3. Conecte un router o un puerto Ethernet conectado a Internet al puerto Ethernet y USB-Host de su placa MicroZed.

Configure el entorno de desarrollo.

Para realizar configuraciones de FreeRTOS para el kit de MicroZed, debe utilizar el Kit de desarrollo de software de Xilinx (XSDK). XSDK es compatible con Windows y Linux.

Descargar e instalar XSDK

Para instalar el software Xilinx, necesita una cuenta gratuita de Xilinx.

Para descargar XSDK

1. Vaya a la página de descarga de Software Development Kit Standalone WebInstall Client.
2. Elija la opción apropiada para su sistema operativo.
3. Se le dirigirá a la página de inicio de sesión de Xilinx.

   Si tiene una cuenta con Xilinx, introduzca sus credenciales de inicio de sesión y elija Iniciar sesión.

   Si no tiene una cuenta de, elija Create your account (Crear cuenta). Después de registrarse, debería recibir un correo electrónico con un enlace para activar su cuenta de Xilinx.
4. En la página Name and Address Verification (Verificación de nombre y dirección), escriba su información y elija Next (Siguiente). La descarga debe estar lista para iniciarse.
5. Guarde el archivo `Xilinx_SDK_`*`version_os`*.

Para instalar XSDK

1. Abra el archivo `Xilinx_SDK_`*`version_os`*.
2. En Select Edition to Install (Seleccionar edición que instalar), elija Xilinx Software Development Kit (XSDK) (Kit de desarrollo de software de Xilinx (XSDK)) y, a continuación, haga clic en Next (Siguiente).

3. En la página siguiente del asistente de instalación, en Installation Options (Opciones de instalación), seleccione Install Cable Drivers (Instalar controladores de cable) y haga clic en Next (Siguiente).

Si su equipo no detecta la conexión del USB-UART de MicroZed, instale manualmente los controladores CP210x de USB a VCP de puente UART. Para obtener instrucciones, consulte Silicon Labs CP210x USB-to-UART Installation Guide.

Para obtener más información acerca del XSDK, consulte Getting Started with Xilinx SDK en el sitio web de Xilink.

Monitorización de mensajes de MQTT en la nube

Antes de ejecutar el proyecto de demostración de FreeRTOS, puede configurar el cliente de MQTT en la consola de AWS IoT para monitorizar los mensajes que envía el dispositivo a la nube de AWS.

Para suscribirse al tema de MQTT con el cliente de MQTT de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***your-thing-name*****/example/topic** y, a continuación, elija Suscribirse al tema.

Creación y ejecución del proyecto de demostración de FreeRTOS

Apertura de la demostración de FreeRTOS en el IDE de XSDK

1. Lance el IDE de XSDK con el directorio del espacio de trabajo establecido en *freertos*`/projects/xilinx/microzed/xsdk`.
2. Cierre la página de bienvenida. En el menú, elija Project (Proyecto) y desactive Build Automatically (Compilar automáticamente).
3. En el menú, elija File (Archivos) y, a continuación, elija Import (Importar).
4. En la página Select (Seleccionar), expanda General, elija Existing Projects into Workspace (Proyectos existentes en Workspace) y, a continuación, elija Next (Siguiente).

5. En la página Importar proyectos, elija Seleccionar directorio raíz y escriba el directorio raíz del proyecto de demostración: *freertos*/projects/xilinx/microzed/xsdk/aws_demos. Para examinar el directorio, haga clic en Browse (Examinar).

   Después de especificar un directorio raíz, los proyectos de dicho directorio aparecen en la página Import Projects (Importar proyectos). Todos los proyectos disponibles se seleccionan de forma predeterminada.

   > **Note**
   >
   > Si ve una advertencia en la parte superior de la página Import Projects (Importar proyectos) ("Algunos proyectos no se puede importar, porque ya existen en el espacio de trabajo") puede ignorarla.

6. Con todos los proyectos seleccionados, elija Finish (Finalizar).
7. Si no ve los proyectos aws_bsp, fsbl y MicroZed_hw_platform_0 en el panel de proyectos, repita los pasos anteriores empezando por el número 3 pero con el directorio raíz establecido en *freertos*/vendors/xilinx e importe aws_bsp, fsbl y MicroZed_hw_platform_0.
8. En el menú, elija Window (Ventana) y, a continuación, elija Preferences (Preferencias).
9. En el panel de navegación, expanda Run/Debug (Ejecutar/Depurar), elija String Substitution (Sustitución de cadenas) y, elija New (Nueva).
10. En New String Substitution Variable (Nueva variable de sustitución de cadena), en Name (Nombre), escriba **AFR_ROOT**. En Valor, escriba la ruta raíz de *freertos*/projects/xilinx/microzed/xsdk/aws_demos. Seleccione OK (Aceptar) y, a continuación, elija OK (Aceptar) para guardar la variable y cerrar Preferences (Preferencias).

Creación del proyecto de demostración de FreeRTOS

1. En el IDE de XSDK, desde el menú, elija Project (Proyecto) y, a continuación, elija Clean (Limpiar).
2. En Clean (Limpiar), deje las opciones en los valores predeterminados y, a continuación, elija OK (Aceptar). XSDK limpia y compila todos los proyectos y, a continuación, genera los archivos .elf.

> **Note**
>
> Para compilar todos los proyectos sin limpiarlos, elija Project (Proyecto) y, a continuación, elija Build All (Compilar todo).
> Para compilar proyectos individuales, seleccione el proyecto que desea compilar, elija Project (Proyecto) y, a continuación, elija Build Project (Compilar proyecto).

Generación de la imagen de arranque del proyecto de demostración de FreeRTOS

1. En el IDE de XSDK, haga clic con el botón derecho en aws_demos y, a continuación, seleccione Create Boot Image (Crear imagen de arranque).
2. En Create Boot Image (Crear imagen de arranque), elija Create new BIF file (Crear nuevo archivo BIF).
3. Junto a Output BIF file path (Ruta de archivo BIF de salida), elija Browse (Examinar) y, a continuación, elija `aws_demos.bif` ubicado en *<freertos>*`/vendors/xilinx/microzed/aws_demos/aws_demos.bif`.
4. Elija Add (Agregar).
5. En Add new boot image partition (Añadir nueva partición de imagen de inicio), junto a File path (Ruta de archivo), haga clic en Browse (Examinar) y elija `fsbl.elf`, situado en `vendors/xilinx/fsbl/Debug/fsbl.elf`.
6. Para Partition type (Tipo de partición), elija bootloader y, a continuación, elija OK (Aceptar).
7. En Create Boot Image (Crear imagen de arranque), elija Create Image (Crear imagen). En Override Files (Anular archivos), elija OK (Aceptar) para sobrescribir el archivo `aws_demos.bif` existente y generar el archivo `BOOT.bin` en `projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin`.

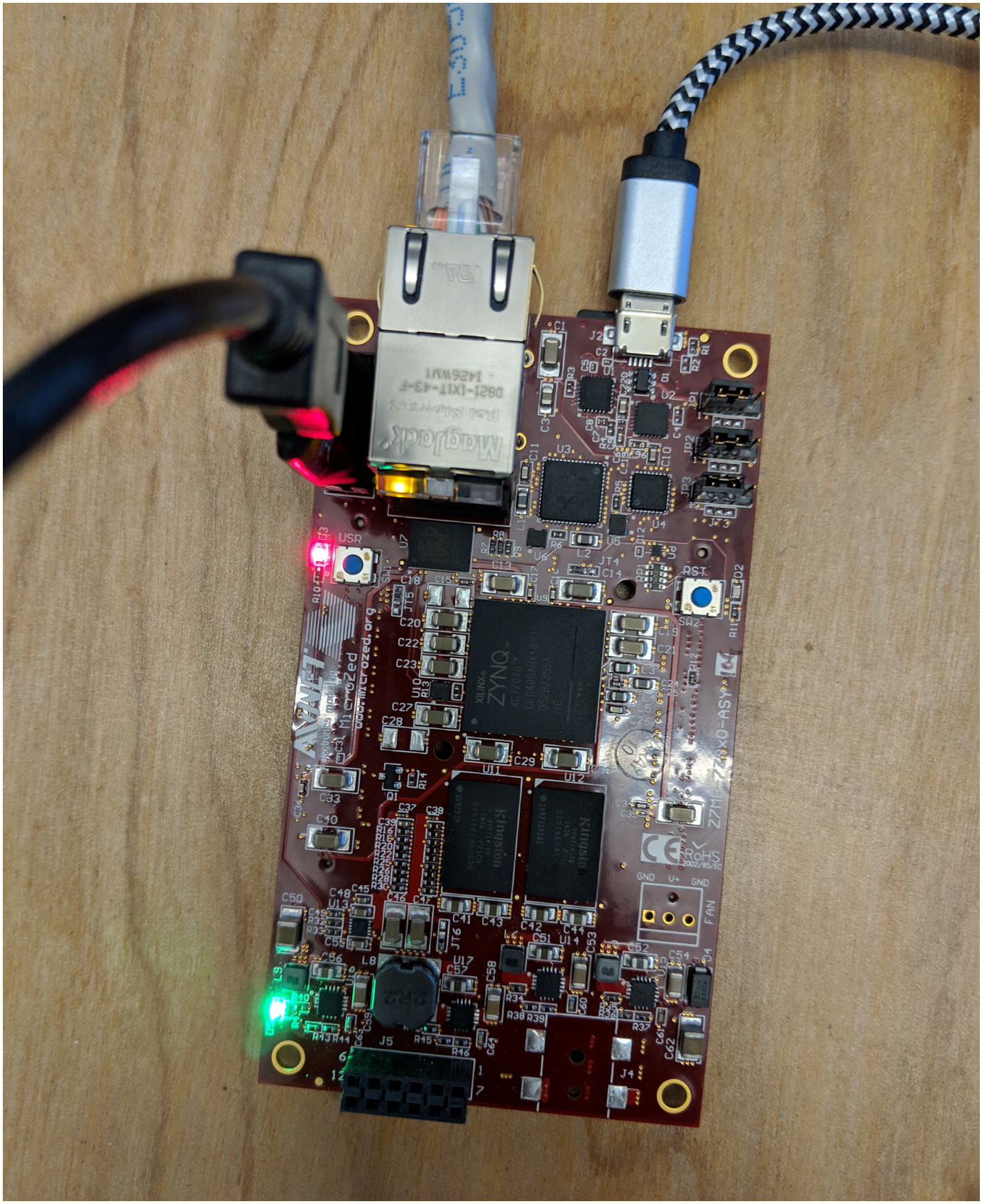
Depuración de JTAG

1. Establezca los puentes del modo de arranque de la placa MicroZed en el modo de arranque JTAG.

2. Inserte su tarjeta MicroSD en la ranura para tarjeta MicroSD ubicado directamente en el puerto USB-UART.

> **Note**
>
> Antes de depurar, asegúrese de hacer una copia de seguridad de cualquier contenido que tiene en la tarjeta MicroSD.

La placa debe ser similar a la siguiente:


USR
RST
ZYNQ
XILINX
Kingston
Kingston
RoHS
GND
V+
GND
FAN

3. En el IDE del XSDK, haga clic con el botón derecho en aws_demos, elija Debug As (Depurar como) y, a continuación, elija 1 Launch on System Hardware (System Debugger) (1 Lanzar en hardware del sistema (Depurador del sistema)).
4. Cuando el depurador se detiene en el punto de ruptura en `main()`, en el menú, elija Run (Ejecutar) y, a continuación, elija Resume (Reanudar).

> **Note**
>
> La primera vez que ejecute la aplicación, se importa un nuevo par de certificado-clave en la memoria no volátil. Para ejecuciones posteriores, puede marcar como comentario `vDevModeKeyProvisioning()` en el archivo `main.c` antes de recompilar las imágenes y el archivo `BOOT.bin`. Esto impide que la copia de los certificados y la clave se almacenen en cada ejecución.

Puede elegir arrancar su placa MicroZed desde una tarjeta MicroSD o desde la memoria flash QSPI para ejecutar el proyecto de demostración de FreeRTOS. Para obtener instrucciones, consulte Generación de la imagen de arranque del proyecto de demostración de FreeRTOS y Ejecución del proyecto de demostración de FreeRTOS.

Ejecución del proyecto de demostración de FreeRTOS

Para ejecutar el proyecto de demostración de FreeRTOS, puede elegir arrancar su placa MicroZed desde una tarjeta MicroSD o desde la memoria flash QSPI.

A medida que configura su placa MicroZed para ejecutar el proyecto de demostración de FreeRTOS, consulte el diagrama en Configuración de hardware MicroZed. Asegúrese de que ha conectado la placa MicroZed en el equipo.

Arranque del proyecto de FreeRTOS desde una tarjeta MicroSD

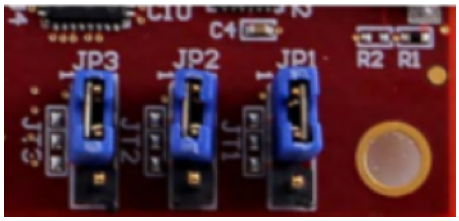
Formatee la tarjeta MicroSD, que se proporciona con Xilinx MicroZed Industrial IoT Kit.

1. Copie el archivo `BOOT.bin` en la tarjeta MicroSD.
2. Inserte la tarjeta en la ranura de la tarjeta MicroSD directamente debajo del puerto USB-UART.
3. Establezca los puentes del modo de arranque de MicroZed en el modo de arranque SD.

4. Pulse el botón RST para restablecer el dispositivo y comenzar a arrancar la aplicación. También puede desenchufar el cable USB-UART del puerto USB-UART y, a continuación, volver a insertar el cable.

Arranque del proyecto de demostración de FreeRTOS desde la memoria flash QSPI

1. Establezca los puentes del modo de arranque de la placa MicroZed en el modo de arranque JTAG.



2. Compruebe que el equipo se encuentra conectado a los puertos de acceso USB-UART y JTAG. El indicador LED verde de alimentación correcta debe iluminarse.
3. En el IDE del XSDK, desde el menú, elija Xilinx y, a continuación, elija Program Flash (Programar flash).
4. En Program Flash Memory (Programar memoria flash), la plataforma de hardware debe completarse de forma automática. Para Connection (Conexión), elija el servidor de hardware MicroZed para conectar la placa con su equipo host.

> **Note**
>
> Si está utilizando el cable Xilinx Smart Lync JTAG, debe crear un servidor de hardware en el IDE de XSDK. Elija New (Nuevo) y, a continuación, defina su servidor.

5. En Image File (Archivo de imagen), escriba la ruta del directorio a su archivo de imagen `BOOT.bin`. Elija Browse (Examinar) para examinar el archivo en su lugar.
6. En Offset (Desplazamiento), escriba **`0x0`**.
7. En FSBL File (Archivo FSBL), escriba la ruta del directorio a su archivo `fsbl.elf`. Elija Browse (Examinar) para examinar el archivo en su lugar.

8. Elija Program (Programar) para programar la placa.
9. Una vez que la programación de QSPI se haya completado, retire el cable USB-UART para apagar la placa.
10. Establezca los puentes del modo de arranque de la placa MicroZed en el modo de arranque QSPI.
11. Inserte su tarjeta en la ranura para tarjeta MicroSD ubicada directamente en el puerto USB-UART.

> **Note**
>
> Asegúrese de hacer una copia de seguridad de cualquier contenido que tenga en la tarjeta MicroSD.

12. Pulse el botón RST para restablecer el dispositivo y comenzar a arrancar la aplicación. También puede desenchufar el cable USB-UART del puerto USB-UART y, a continuación, volver a insertar el cable.

Solución de problemas

Si detecta errores de compilación relacionados con rutas incorrectas, pruebe a limpiar y recompilar el proyecto, como se describe en Creación del proyecto de demostración de FreeRTOS.

Si utiliza Windows, asegúrese de que utiliza barras diagonales al establecer la sustitución en la cadena de variables del IDE del XSDK de Windows.

Si necesita información general de solución de problemas que pueden surgir al empezar a trabajar con FreeRTOS, consulte Introducción a solución de problemas.

Próximos pasos con FreeRTOS

> **Important**
>
> Esta página hace referencia al repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Después de crear, instalar y ejecutar el proyecto de demostración de FreeRTOS para su placa, puede visitar el sitio web FreeRTOS.org para obtener más información sobre cómo Crear un nuevo proyecto de FreeRTOS. También hay demostraciones de muchas bibliotecas de FreeRTOS que muestran cómo realizar tareas importantes, interactuar con los servicios de AWS IoT y programar capacidades específicas de la placa (como los módems móviles). Para obtener más información, consulte la página Categorías de la biblioteca FreeRTOS.

El sitio web FreeRTOS.org también tiene información detallada sobre el kernel de FreeRTOS, así como conceptos fundamentales del sistema operativo en tiempo real. Para obtener más información, consulte las páginas de Documentos para desarrolladores del kernel de FreeRTOS y Documentos secundarios del kernel de FreeRTOS.

Actualizaciones vía inalámbrica de FreeRTOS

> **Note**
>
> Consulte Actualizaciones inalámbricas de AWS IoT en el sitio web de FreeRTOS para obtener información reciente sobre cómo realizar actualizaciones inalámbricas.

Las actualizaciones inalámbricas (OTA) le permiten implementar actualizaciones de firmware en uno o varios dispositivos de su flota. Aunque las actualizaciones OTA están pensadas para actualizar firmware de dispositivos, puede utilizarlas para enviar archivos a uno o más dispositivos registrados con AWS IoT. Cuando envíe actualizaciones inalámbricas, le recomendamos que las firme digitalmente para que los dispositivos que reciben los archivos puedan verificar que no se han manipulado durante el trayecto.

Puede utilizar la Firma de código para AWS IoT para firmar los archivos o puede firmar los archivos con sus propias herramientas de firma de código.

Cuando se crea una actualización OTA, el Servicio OTA Update Manager genera un trabajo de AWS IoT para notificar a los dispositivos que hay una actualización disponible. La aplicación de demostración de OTA se ejecuta en su dispositivo y crea una tarea de FreeRTOS que se suscribe a temas de notificación para trabajos de AWS IoT y escucha mensajes de actualización. Cuando hay una actualización disponible, el agente de OTA publica solicitudes en AWS IoT y recibe actualizaciones mediante el protocolo HTTP o MQTT, dependiendo de la configuración elegida. El agente de OTA comprueba la firma digital de los archivos descargados y si es válida, instala la actualización de firmware. Si no utiliza la aplicación de demostración de actualización OTA de

FreeRTOS, debe integrar la Biblioteca de actualizaciones inalámbricas de AWS IoT en su propia aplicación para obtener la funcionalidad de actualización de firmware.

Las actualizaciones vía inalámbrica de FreeRTOS le permiten:

- Firme digitalmente el firmware antes de la implementación.
- Implementar nuevas imágenes de firmware en un único dispositivo, grupo de dispositivos o toda la flota.
- Implementar firmware en dispositivos a medida que estos se añaden a grupos, restablecen o aprovisionan.
- Verificar la autenticidad y la integridad del nuevo firmware una vez implementado en los dispositivos.
- Monitorizar el progreso de una implementación.
- Depurar una implementación infructuosa.

Etiquetado de recursos de OTA

Para administrar mejor los recursos de OTA, puede asignar opcionalmente sus propios metadatos a las actualizaciones y flujos en forma de etiquetas. Las etiquetas le permiten clasificar los recursos de AWS IoT de diversas maneras (por ejemplo, según su finalidad, propietario o entorno). Esto es útil cuando se tienen muchos recursos del mismo tipo. Puede identificar rápidamente un recurso según las etiquetas que le haya asignado.

Para obtener más información, consulte Etiquetado de los recursos de AWS IoT.

Requisitos previos de actualización OTA

Para utilizar actualizaciones inalámbricas (OTA), haga lo siguiente:

- Compruebe el Requisitos previos para las actualizaciones de OTA mediante HTTP o el Requisitos previos para las actualizaciones de OTA mediante MQTT.
- Creación de un bucket de Amazon S3 para almacenar la actualización.
- Crear un rol de servicio de actualizaciones OTA.
- Crear una política de usuario de OTA.
- Crear un certificado de firma de código.
- Si utiliza Code Signing para AWS IoT, Conceder acceso a Code Signing para AWS IoT.
- Descarga de FreeRTOS con la biblioteca de OTA.

Creación de un bucket de Amazon S3 para almacenar la actualización

Los archivos de actualización OTA se almacenan en buckets de Amazon S3.

Si utiliza Code Signing para AWS IoT, el comando que utiliza para crear un trabajo de firma de código necesita un bucket de origen (donde se encuentra la imagen de firmware) y un bucket de destino (donde se escribe la imagen de firmware firmada). Puede especificar el mismo bucket para el origen y el destino. Los nombres de los archivos cambian a GUID para no sobrescribir los archivos originales.

Para crear un bucket de Amazon S3

1. Inicie sesión en la consola de Amazon S3 en https://console.aws.amazon.com/s3/.
2. Elija Crear bucket.
3. Escriba un nombre de bucket.
4. En Configuración del bucket para Bloquear acceso público, mantenga seleccionada la opción Bloquear todo el acceso público para aceptar los permisos predeterminados.
5. En Control de versiones de buckets, seleccione Habilitar para conservar todas las versiones en el mismo bucket.
6. Elija Crear bucket.

Para obtener más información acerca de Amazon S3, consulte la Guía del usuario de Amazon Simple Storage Service.

Crear un rol de servicio de actualizaciones OTA

El servicio de actualizaciones OTA toma este rol para crear y administrar trabajos de actualización OTA en su nombre.

Para crear un rol de servicio de OTA

1. Inicie sesión en https://console.aws.amazon.com/iam/.
2. En el panel de navegación, elija Roles.
3. Elija Create role (Crear rol).
4. En Select type of trusted entity (Seleccionar el tipo de entidad de confianza), elija AWS Service (Servicio de AWS).
5. Elija IoT en la lista de servicios de AWS.

6. En Select your use case (Seleccione su caso de uso), elija IoT.
7. Elija Next: Permissions.
8. Elija Next: Tags (Siguiente: Etiquetas).
9. Elija Next: Review (Siguiente: revisar).
10. Introduzca un nombre y una descripción del rol y, a continuación, elija Create rol (Crear rol).

Para obtener más información sobre los roles de IAM, consulte Roles de IAM.

> **Important**
>
> Para abordar el confuso problema de los agentes de seguridad, debe seguir las instrucciones de la guía de AWS IoT Core.

Para añadir permisos de actualización OTA a su rol de servicio de OTA

1. En el campo de búsqueda de la página de la consola de IAM, escriba el nombre del rol y, a continuación, selecciónelo de la lista.
2. Seleccione Attach policies (Asociar políticas).
3. En el cuadro Search (Buscar), escriba "AmazonFreeRTOSOTAUpdate", seleccione AmazonFreeRTOSOTAUpdate en la lista de políticas filtradas y, a continuación, elija Attach policy (Asociar política) para asociar la política al rol de servicio.

Para añadir los permisos de IAM necesarios a su rol de servicio de OTA

1. En el campo de búsqueda de la página de la consola de IAM, escriba el nombre del rol y, a continuación, selecciónelo de la lista.
2. Elija Add inline policy (Agregar política insertada).
3. Seleccione la pestaña JSON.
4. Copie y pegue el siguiente documento de política en el cuadro de texto:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
```

```
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::your_account_id:role/your_role_name"
        }
    ]
}
```

No olvide sustituir *your_account_id* por el ID de la cuenta de AWS y *your_role_name* por el nombre del rol del servicio de OTA.

5. Elija Review policy (Revisar política).
6. Escriba un nombre para la política y elija Create policy (Crear política).

> **Note**
>
> El procedimiento siguiente no es necesario si el nombre del bucket de Amazon S3 comienza por “afr-ota”. Si lo hace, la política administrada por AWS `AmazonFreeRTOSOTAUpdate` ya incluye los permisos necesarios.

Para añadir los permisos de Amazon S3 necesarios a su rol de servicio de OTA

1. En el campo de búsqueda de la página de la consola de IAM, escriba el nombre del rol y, a continuación, selecciónelo de la lista.
2. Elija Add inline policy (Agregar política insertada).
3. Seleccione la pestaña JSON.
4. Copie y pegue el siguiente documento de política en el cuadro.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetObject",
```

```
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
            ]
        }
    ]
}
```

Esta política concede a su rol de servicio de OTA permisos para leer objetos de Amazon S3. No olvide sustituir *example-bucket* por el nombre del bucket.

5. Elija Review policy (Revisar política).
6. Escriba un nombre para la política y elija Create policy (Crear política).

Crear una política de usuario de OTA

Debe conceder a su usuario de permiso para realizar actualizaciones inalámbricas. Su usuario debe tener permisos para:

- Acceder al bucket de S3 donde se almacenan las actualizaciones de firmware.
- Acceder a los certificados almacenados en AWS Certificate Manager.
- Acceder a la característica de entrega de archivos basada en MQTT de AWS IoT.
- Acceder a las actualizaciones OTA de FreeRTOS.
- Acceder a los trabajos de AWS IoT.
- Acceder a IAM.
- Obtenga acceso a Code Signing para AWS IoT. Consulte Conceder acceso a Code Signing para AWS IoT.
- Enumerar las plataformas de certificación de FreeRTOS.
- Etiquetar y desetiquetar recursos de AWS IoT.

Para otorgar los permisos necesarios al usuario, consulte Políticas de IAM. Consulte también Autorización de los usuarios y los servicios en la nube para usar Trabajos de AWS IoT.

Para proporcionar acceso, agregue permisos a sus usuarios, grupos o roles:

- Usuarios y grupos de AWS IAM Identity Center:

Cree un conjunto de permisos. Siga las instrucciones de Create a permission set (Creación de un conjunto de permisos) en la Guía del usuario de AWS IAM Identity Center.

- Usuarios administrados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones de Creación de un rol para un proveedor de identidad de terceros (federación) en la Guía del usuario de IAM.

- Usuarios de IAM:
  - Cree un rol que el usuario pueda asumir. Siga las instrucciones de Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.
  - (No recomendado) Adjunte una política directamente a un usuario o agregue un usuario a un grupo de usuarios. Siga las instrucciones de Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Crear un certificado de firma de código

Para firmar digitalmente las imágenes de firmware, necesita un certificado de firma de código y una clave privada. Para realizar pruebas, puede crear un certificado autofirmado y una clave privada. Para entornos de producción, debe adquirir un certificado firmado a través de una entidad de certificación (CA) conocida.

Las distintas plataformas requieren diferentes tipos de certificados de firma de código. En las siguientes secciones se describe cómo crear certificados de firma de código para diferentes plataformas calificadas para FreeRTOS.

Temas

- Creación de un certificado de firma de código para CC3220SF-LAUNCHXL de Texas Instruments
- Creación de un certificado de firma de código para ESP32 de Espressif
- Creación de un certificado de firma de código para Nordic nrf52840-dk
- Creación de un certificado de firma de código para el simulador de Windows de FreeRTOS
- Creación de un certificado de firma de código para hardware personalizado

Creación de un certificado de firma de código para CC3220SF-LAUNCHXL de Texas Instruments

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

El SimpleLink Wi-Fi CC3220SF Wireless Microcontroller Launchpad Development Kit admite dos cadenas de certificados para la firma de código de firmware:

- Producción (certificado-catalogo)

  Para utilizar la cadena de certificados de producción, debe comprar un certificado de firma de código comercial y utilizar la herramienta Uniflash de TI para colocar la placa en el modo de producción.

- Pruebas y desarrollo (certificado-sitio de pruebas)

  La cadena de certificados de sitio de pruebas le permite probar actualizaciones OTA con un certificado de firma de código autofirmado.

Utilice la AWS Command Line Interface para importar el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager. Para obtener más información, consulte Instalación de la AWS CLI en la Guía del usuario de AWS Command Line Interface.

Descargue e instale la versión más reciente del SDK de SimpleLink CC3220. De forma predeterminada, los archivos que necesita se encuentran aquí:

`C:\ti\simplelink_cc32xx_sdk_`*`version`*`\tools\cc32xx_tools\certificate-playground` (Windows)

`/Applications/Ti/simplelink_cc32xx_`*`version`*`/tools/cc32xx_tools/certificate-playground` (macOS)

Los certificados en el SimpleLink CC3220 SDK están en formato DER. Para crear un certificado de firma de código autofirmado, debe convertirlo a formato PEM.

Siga estos pasos para crear un certificado de firma de código que esté vinculado a la jerarquía de certificados del sitio de pruebas de Texas Instruments y cumpla los criterios de AWS Certificate Manager y Code Signing para AWS IoT.

> **Note**
>
> Para crear un certificado de firma de código, instale OpenSSL en su equipo. Después de instalar OpenSSL, asegúrese de que `openssl` se asigna al ejecutable de OpenSSL en el símbolo del sistema o el terminal entorno.

Para crear un certificado de firma de código autofirmado

1. Abra un símbolo del sistema o terminal con permisos de administrador.
2. En el directorio de trabajo, use el siguiente texto para crear un archivo llamado `cert_config.txt`. Sustituya *`test_signer@amazon.com`* con su dirección de correo electrónico.

```
[ req ]
prompt             = no
distinguished_name = my dn

[ my dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

3. Cree una clave privada y una solicitud de firma de certificado (CSR):

```
openssl req -config cert_config.txt -extensions my_exts -nodes -days 365 -newkey
 rsa:2048 -keyout tisigner.key -out tisigner.csr
```

4. Convierta la clave privada de CA de la raíz de sitio de pruebas de Texas Instruments del formato DER al formato PEM.

   La clave privada de CA de raíz de sitio de pruebas de TI se encuentra aquí:

`C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground\dummy-root-ca-cert-key` (Windows)

`/Applications/Ti/simplelink_cc32xx_sdk_version/tools/cc32xx_tools/certificate-playground/dummy-root-ca-cert-key` (macOS)

```
openssl rsa -inform DER -in dummy-root-ca-cert-key -out dummy-root-ca-cert-key.pem
```

5. Convierta el certificado de CA de raíz de sitio de pruebas de Texas Instruments del formato DER al formato PEM.

   La clave privada del certificado de raíz de sitio de pruebas de TI se encuentra aquí:

   `C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground/dummy-root-ca-cert` (Windows)

   `/Applications/Ti/simplelink_cc32xx_sdk_version/tools/cc32xx_tools/certificate-playground/dummy-root-ca-cert` (macOS)

   ```
   openssl x509 -inform DER -in dummy-root-ca-cert -out dummy-root-ca-cert.pem
   ```

6. Firme la CSR con la CA raíz de Texas Instruments:

   ```
   openssl x509 -extfile cert_config.txt -extensions my_exts  -req -days 365 -in
    tisigner.csr -CA dummy-root-ca-cert.pem -CAkey dummy-root-ca-cert-key.pem -
   set_serial 01 -out tisigner.crt.pem -sha1
   ```

7. Convierta su certificado de firma de código (`tisigner.crt.pem`) a formato DER:

   ```
   openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
   ```

   > **Note**
   >
   > Puede escribir el certificado `tisigner.crt.der` en la placa de desarrollo de TI más tarde.

8. Importe el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager:

```
aws acm import-certificate --certificate fileb://tisigner.crt.pem --private-key
 fileb://tisigner.key --certificate-chain fileb://dummy-root-ca-cert.pem
```

Este comando muestra un ARN para su certificado. Necesita este ARN al crear un trabajo de actualización OTA.

> **Note**
>
> Este paso se ha escrito partiendo del supuesto de que va a utilizar Code Signing para AWS IoT para firmar las imágenes de firmware. Aunque se recomienda usar Code Signing para AWS IoT, puede firmar las imágenes de firmware manualmente.

Creación de un certificado de firma de código para ESP32 de Espressif

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Las placas ESP32 de Espressif admiten SHA256 autofirmado con certificados de firma de código ECDSA.

> **Note**
>
> Para crear un certificado de firma de código, instale OpenSSL en su equipo. Después de instalar OpenSSL, asegúrese de que `openssl` se asigna al ejecutable de OpenSSL en el símbolo del sistema o el terminal entorno.
> Utilice la AWS Command Line Interface para importar el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager. Para obtener información acerca de cómo instalar la AWS CLI, consulte Instalación de la AWS CLI.

1. En el directorio de trabajo, use el siguiente texto para crear un archivo llamado `cert_config.txt`. Sustituya *`test_signer@amazon.com`* con su dirección de correo electrónico:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Cree una clave privada de firma de código ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Cree un certificado de firma de código ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importe el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager:

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Este comando muestra un ARN para su certificado. Necesita este ARN al crear un trabajo de actualización OTA.

> **Note**
>
> Este paso se ha escrito partiendo del supuesto de que va a utilizar Code Signing para AWS IoT para firmar las imágenes de firmware. Aunque se recomienda usar Code Signing para AWS IoT, puede firmar las imágenes de firmware manualmente.

Creación de un certificado de firma de código para Nordic nrf52840-dk

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Nordic nrf52840-dk admite SHA256 auto-firmado con certificados de firma de código ECDSA.

> **Note**
>
> Para crear un certificado de firma de código, instale OpenSSL en su equipo. Después de instalar OpenSSL, asegúrese de que `openssl` se asigna al ejecutable de OpenSSL en el símbolo del sistema o el terminal entorno.
> Utilice la AWS Command Line Interface para importar el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager. Para obtener información acerca de cómo instalar la AWS CLI, consulte Instalación de la AWS CLI.

1. En el directorio de trabajo, use el siguiente texto para crear un archivo llamado `cert_config.txt`. Sustituya *`test_signer@amazon.com`* con su dirección de correo electrónico:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Cree una clave privada de firma de código ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Cree un certificado de firma de código ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importe el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager:

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

   Este comando muestra un ARN para su certificado. Necesita este ARN al crear un trabajo de actualización OTA.

   > **Note**
   >
   > Este paso se ha escrito partiendo del supuesto de que va a utilizar Code Signing para AWS IoT para firmar las imágenes de firmware. Aunque se recomienda usar Code Signing para AWS IoT, puede firmar las imágenes de firmware manualmente.

Creación de un certificado de firma de código para el simulador de Windows de FreeRTOS

El simulador de Windows de FreeRTOS requiere un certificado de firma de código con una clave ECDSA P-256 y hash SHA-256 para realizar actualizaciones OTA. Si no dispone de un certificado de firma de código, siga estos pasos para crear uno.

> **Note**
>
> Para crear un certificado de firma de código, instale OpenSSL en su equipo. Después de instalar OpenSSL, asegúrese de que `openssl` se asigna al ejecutable de OpenSSL en el símbolo del sistema o el terminal entorno.
> Utilice la AWS Command Line Interface para importar el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager. Para obtener información acerca de cómo instalar la AWS CLI, consulte Instalación de la AWS CLI.

1. En el directorio de trabajo, use el siguiente texto para crear un archivo llamado `cert_config.txt`. Sustituya *`test_signer@amazon.com`* con su dirección de correo electrónico:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Cree una clave privada de firma de código ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Cree un certificado de firma de código ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importe el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager:

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Este comando muestra un ARN para su certificado. Necesita este ARN al crear un trabajo de actualización OTA.

> **Note**
>
> Este paso se ha escrito partiendo del supuesto de que va a utilizar Code Signing para AWS IoT para firmar las imágenes de firmware. Aunque se recomienda usar Code Signing para AWS IoT, puede firmar las imágenes de firmware manualmente.

Creación de un certificado de firma de código para hardware personalizado

Con el conjunto de herramientas adecuado, cree una clave privada y un certificado auto-firmado para su hardware.

Utilice la AWS Command Line Interface para importar el certificado de firma de código, la clave privada y la cadena de certificados a AWS Certificate Manager. Para obtener información acerca de cómo instalar la AWS CLI, consulte Instalación de la AWS CLI.

Después de crear su certificado de firma de código, puede utilizar la AWS CLI para importarlo en ACM:

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://
code-sign.key
```

La salida de este comando muestra un ARN para su certificado. Necesita este ARN al crear un trabajo de actualización OTA.

ACM requiere que los certificados utilicen algoritmos y tamaños de claves específicos. Para obtener más información, consulte Requisitos previos para la importación de certificados. Para obtener más información acerca de ACM, consulte Importación de certificados a AWS Certificate Manager.

Debe copiar, pegar y formatear el contenido de su certificado de firma de código en el archivo `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` que forma parte del código de FreeRTOS que descargará más adelante.

Conceder acceso a Code Signing para AWS IoT

Para proporcionar acceso, agregue permisos a sus usuarios, grupos o roles:

- Usuarios y grupos de AWS IAM Identity Center:

  Cree un conjunto de permisos. Siga las instrucciones de Create a permission set (Creación de un conjunto de permisos) en la Guía del usuario de AWS IAM Identity Center.
- Usuarios administrados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones de Creación de un rol para un proveedor de identidad de terceros (federación) en la Guía del usuario de IAM.
- Usuarios de IAM:
  - Cree un rol que el usuario pueda asumir. Siga las instrucciones de Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.

- (No recomendado) Adjunte una política directamente a un usuario o agregue un usuario a un grupo de usuarios. Siga las instrucciones de Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Descarga de FreeRTOS con la biblioteca de OTA

Puede clonar o descargar FreeRTOS desde GitHub. Consulte el archivo README.md para obtener instrucciones.

Para obtener más información acerca de cómo configurar y ejecutar la aplicación de demostración de OTA, consulte Aplicación de demostración de actualizaciones transparentes.

Important

- En este tema, la ruta al directorio de descargas de FreeRTOS se denomina *freertos*.
- Los caracteres de espacio en la ruta *freertos* pueden causar errores de compilación. Cuando clone o copie el repositorio, asegúrese de que la ruta que crea no contiene caracteres de espacio.
- La longitud máxima de una ruta de archivo en Microsoft Windows es de 260 caracteres. Las rutas largas al directorio de descargas de FreeRTOS pueden provocar errores de creación.
- Como el código fuente puede contener enlaces simbólicos, si utiliza Windows para extraer el archivo, es posible que tenga que:
  - Habilitar el modo de desarrollador o
  - Utilizar una consola que tenga el rango de administrador.

  De esta forma, Windows puede crear correctamente enlaces simbólicos al extraer el archivo. De lo contrario, los enlaces simbólicos se escribirán como archivos normales que contengan las rutas de los enlaces simbólicos como texto o estarán vacíos. Para obtener más información, consulte la entrada del blog Symlinks in Windows 10 .

  Si usa Git en Windows, debe habilitar el modo desarrollador o debe:

  - Establecer `core.symlinks` en verdadero con el siguiente comando:

```
git config --global core.symlinks true
```

- Usar una consola que tenga el rango de administrador siempre que utilice un comando git que escriba en el sistema (por ejemplogit pull, git clone y git submodule update --init --recursive).

Requisitos previos para las actualizaciones de OTA mediante MQTT

En esta sección se describen los requisitos generales para utilizar MQTT para realizar actualizaciones inalámbricas (OTA).

Requisitos mínimos

- El firmware del dispositivo debe incluir las bibliotecas de FreeRTOS necesarias (agente de coreMQTT, actualización OTA y sus dependencias).
- Se necesita la versión 1.4.0 o posterior de FreeRTOS. Sin embargo, le recomendamos que utilice la versión más reciente cuando sea posible.

Configuraciones

A partir de la versión 201912.00, OTA de FreeRTOS puede utilizar el protocolo HTTP o MQTT para transferir imágenes de actualización de firmware desde AWS IoT a dispositivos. Si especifica ambos protocolos al crear una actualización OTA en FreeRTOS, cada dispositivo determinará el protocolo utilizado para transferir la imagen. Para obtener más información, consulte Requisitos previos para las actualizaciones de OTA mediante HTTP.

De manera predeterminada, la configuración de los protocolos OTA en `ota_config.h` es utilizar el protocolo MQTT.

Configuraciones específicas del dispositivo

Ninguno.

Uso de memoria

Cuando se utiliza MQTT para la transferencia de datos, no se requiere memoria adicional para la conexión MQTT porque se comparte entre operaciones de control y datos.

Política de dispositivos

Cada dispositivo que reciba una actualización OTA utilizando MQTT debe estar registrado como una cosa AWS IoT y debe tener una política adjunta como la que se muestra aquí. Puede encontrar más

información acerca de los elementos de los objetos `"Resource"` y `"Action"` en las Acciones de la política principal de AWS IoT y en los recursos de acciones principales de AWS IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:partition:iot:region:account:client/
${iot:Connection.Thing.ThingName}"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

Notas

- Los permisos `iot:Connect` permiten que su dispositivo se conecte a AWS IoT a través de MQTT.

- Los permisos `iot:Subscribe` y `iot:Publish` en los temas de trabajos de AWS IoT (`.../jobs/*`) permiten que el dispositivo conectado reciba notificaciones de trabajo y documentos de trabajo, y publique el estado de finalización de una ejecución de trabajo.
- Los permisos `iot:Subscribe` y `iot:Publish` en los temas de flujos de OTA de AWS IoT (`.../streams/*`) permiten que el dispositivo conectado obtenga datos de actualización de OTA desde AWS IoT. Estos permisos son necesarios para realizar actualizaciones de firmware sobre MQTT.
- Los permisos `iot:Receive` permiten a AWS IoT Core publicar mensajes sobre esos temas en el dispositivo conectado. Este permiso se verifica en cada entrega de un mensaje MQTT. Puede utilizar este permiso para revocar el acceso a los clientes que están actualmente suscritos a un tema.

Requisitos previos para las actualizaciones de OTA mediante HTTP

En esta sección se describen los requisitos generales para utilizar HTTP para realizar actualizaciones inalámbricas (OTA). A partir de la versión 201912.00, OTA de FreeRTOS puede utilizar el protocolo HTTP o MQTT para transferir imágenes de actualización de firmware desde AWS IoT a dispositivos.

> **Note**
>
> - Aunque el protocolo HTTP puede utilizarse para transferir la imagen de firmware, la biblioteca de agente coreMQTT sigue siendo necesaria porque otras interacciones con AWS IoT Core utilizan esta biblioteca, incluido el envío o recepción de notificaciones de ejecución de trabajos, documentos de trabajo y actualizaciones de estado de ejecución.
> - Cuando se especifican los protocolos MQTT y HTTP para el trabajo de actualización de OTA, la configuración del software del Agente OTA en cada dispositivo determina el protocolo utilizado para transferir la imagen de firmware. Para cambiar el agente OTA del método de protocolo MQTT predeterminado al protocolo HTTP, puede modificar los archivos de encabezado utilizados para compilar el código fuente de FreeRTOS para el dispositivo.

Requisitos mínimos

- El firmware del dispositivo debe incluir las bibliotecas de FreeRTOS necesarias (agente coreMQTT, HTTP, Agente de OTA y sus dependencias).
- Se requiere la versión 201912.00 o posterior de FreeRTOS para cambiar la configuración de los protocolos de OTA para habilitar la transferencia de datos de OTA a través de HTTP.

Configuraciones

Consulte la siguiente configuración de los protocolos OTA en el archivo \vendors\boards*board*\aws_demos\config_files\ota_config.h.

Para habilitar la transferencia de datos de OTA a través de HTTP

1. Cambie `configENABLED_DATA_PROTOCOLS` a `OTA_DATA_OVER_HTTP`.
2. En las actualizaciones OTA, puede especificar ambos protocolos para que se pueda utilizar el protocolo MQTT o HTTP. Puede establecer el protocolo principal utilizado por el dispositivo en HTTP cambiando `configOTA_PRIMARY_DATA_PROTOCOL` por `OTA_DATA_OVER_HTTP`.

> **Note**
>
> HTTP solo se admite para operaciones de datos de OTA. Para operaciones de control, debe utilizar MQTT.

Configuraciones específicas del dispositivo

ESP32

Debido a una cantidad limitada de RAM, debe desactivar BLE cuando habilite HTTP como protocolo de datos OTA. En el archivo vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h, cambie `configENABLED_NETWORKS` por `AWSIOT_NETWORK_TYPE_WIFI` solamente.

```
/**
     * @brief Configuration flag which is used to enable one or more network
 interfaces for a board.
     *
```

```
    * The configuration can be changed any time to keep one or more network enabled
or disabled.
    * More than one network interfaces can be enabled by using 'OR' operation with
flags for
    * each network types supported. Flags for all supported network types can be
found
    * in "aws_iot_network.h"
    *
    */
   #define configENABLED_NETWORKS      ( AWSIOT_NETWORK_TYPE_WIFI )
```

Uso de memoria

Cuando se utiliza MQTT para la transferencia de datos, no se requiere memoria de montón adicional para la conexión MQTT porque se comparte entre operaciones de control y datos. Sin embargo, para habilitar los datos a través de HTTP se requiere memoria de montón adicional. A continuación, se muestran los datos de uso de memoria de montón para todas las plataformas compatibles, calculados mediante la API `xPortGetFreeHeapSize` de FreeRTOS. Debe asegurarse de que hay suficiente RAM para usar la biblioteca OTA.

CC3220SF-LAUNCHXL de Texas Instruments

- Operaciones de control (MQTT): 12 KB
- Operaciones de datos (HTTP): 10 KB

> **Note**
>
> TI usa mucha menos memoria RAM porque aplica SSL en el hardware, por lo que no usa la biblioteca mbedtls.

Curiosity PIC32MZEF de Microchip

- Operaciones de control (MQTT): 65 KB
- Operaciones de datos (HTTP): 43 KB

Espressif ESP32

- Operaciones de control (MQTT): 65 KB

Operaciones de datos (HTTP): 45 KB

> **Note**
>
> BLE en ESP32 usa alrededor de 87 KB de RAM. No hay suficiente RAM para habilitarlos todos, lo que se menciona en las configuraciones específicas del dispositivo anteriores.

Simulador de Windows

Operaciones de control (MQTT): 82 KB

Operaciones de datos (HTTP): 63 KB

Nordic nrf52840-dk

No se admite HTTP.

Política de dispositivos

Esta política le permite utilizar MQTT o HTTP para las actualizaciones de OTA.

Cada dispositivo que reciba una actualización OTA utilizando HTTP debe estar registrado como una cosa en AWS IoT y debe tener una política adjunta como la que se muestra aquí. Puede encontrar más información acerca de los elementos de los objetos `"Resource"` y `"Action"` en las Acciones de la política principal de AWS IoT y en los recursos de acciones principales de AWS IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:partition:iot:region:account:client/
${iot:Connection.Thing.ThingName}"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

Notas

- Los permisos `iot:Connect` permiten que su dispositivo se conecte a AWS IoT a través de MQTT.
- Los permisos `iot:Subscribe` y `iot:Publish` en los temas de trabajos de AWS IoT (`.../jobs/*`) permiten que el dispositivo conectado reciba notificaciones de trabajo y documentos de trabajo, y publique el estado de finalización de una ejecución de trabajo.
- Los permisos `iot:Receive` permiten a AWS IoT Core publicar mensajes sobre esos temas en el dispositivo conectado actual. Este permiso se verifica en cada entrega de un mensaje MQTT. Puede utilizar este permiso para revocar el acceso a los clientes que están actualmente suscritos a un tema.

Tutorial de OTA

Esta sección contiene un tutorial para actualizar firmware en dispositivos que ejecutan FreeRTOS mediante actualizaciones OTA. Además de las imágenes de firmware, puede usar una actualización OTA para enviar cualquier tipo de archivo a un dispositivo conectado a AWS IoT.

Puede usar la consola de AWS IoT o la AWS CLI para crear una actualización OTA. La consola es la manera más sencilla de comenzar a usar OTA, ya que se encarga de gran parte del trabajo en su nombre. La AWS CLI es útil cuando se automatizan trabajos de actualización OTA, cuando se trabaja con un gran número de dispositivos o cuando se utilizan dispositivos que no se han calificado para FreeRTOS. Para obtener más información sobre cómo calificar dispositivos para FreeRTOS, consulte el sitio web de Socios de FreeRTOS.

Para crear una actualización OTA

1. Implemente una versión inicial del firmware en uno o varios dispositivos.
2. Compruebe que el firmware se ejecuta correctamente.
3. Cuando se requiere una actualización de firmware, realice los cambios en el código y crear la nueva imagen.
4. Si está firmando el firmware manualmente, inicie sesión y, a continuación, cargue la imagen de firmware firmada a su bucket de Amazon S3. Si utiliza firma de código para AWS IoT, cargue la imagen de firmware sin firmar en un bucket de Amazon S3.
5. Cree una actualización OTA.

Cuando se crea una actualización OTA, se especifica el protocolo de entrega de imágenes (MQTT o HTTP) o se especifican ambos para permitir que el dispositivo elija. El agente de OTA de FreeRTOS en el dispositivo recibe la imagen de firmware actualizada y verifica la firma digital, la suma de comprobación y el número de versión de la nueva imagen. Si la actualización de firmware supera la verificación, el dispositivo se restablece y, según la lógica definida por la aplicación, se confirma la actualización. Si sus dispositivos no ejecutan FreeRTOS, debe implementar un agente de OTA que se ejecute en sus dispositivos.

Instalación del firmware inicial

Para actualizar el firmware, debe instalar una versión inicial del firmware que use la biblioteca de agente de OTA para permanecer a la escucha de trabajos de actualización OTA. Si no va a ejecutar FreeRTOS, omita este paso. En cambio, copie su implementación de agente de OTA en sus dispositivos.

Temas

- Instalación de la versión inicial de firmware en CC3220SF-LAUNCHXL de Texas Instruments
- Instalación de la versión inicial de firmware en ESP32 de Espressif
- Instalación de la versión inicial de firmware en Nordic nRF52840 DK
- Firmware inicial en el simulador de Windows
- Instalación de la versión inicial de firmware en una placa personalizada

Instalación de la versión inicial de firmware en CC3220SF-LAUNCHXL de Texas Instruments

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Estos pasos se han redactado partiendo del supuesto de que ya ha creado el proyecto `aws_demos`, como se describe en Descargue, compile, instale y ejecute la demostración de OTA de FreeRTOS en CC3220SF-LAUNCHXL de Texas Instruments.

1. En su CC3220SF-LAUNCHXL de Texas Instruments, coloque el puente SOP en el centro del conjunto de pines (posición = 1) y restablezca la placa.
2. Descargue e instale la herramienta Uniflash de TI.
3. Comience Uniflash. En la lista de configuraciones, elija CC3220SF-LAUNCHXL y, a continuación, elija Start Image Creator (Iniciar creador de imagen).
4. Elija New Project (Nuevo proyecto).
5. En la página Start new project (Comenzar proyecto nuevo), escriba un nombre para el proyecto. En Device Type (Tipo de dispositivo), seleccione CC3220SF. En Device Mode (Modo del dispositivo), elija Develop (Desarrollo). Elija Create Project (Crear proyecto).
6. Desconecte su emulador de terminal.
7. En el lado derecho de la ventana de la aplicación Uniflash, elija Connect (Conectar).
8. En Advanced (Avanzado), Files (Archivos), seleccione User Files (Archivos de usuario).
9. En el panel selector File (Archivo), elija el icono Add File (Añadir archivo)

   .
10. Vaya al directorio `/Applications/Ti/simplelink_cc32xx_sdk_`*`version`*`/tools/cc32xx_tools/certificate-playground`, seleccione `dummy-root-ca-cert`, elija Open (Abrir) y, a continuación, elija Write (Escribir).
11. En el panel selector File (Archivo), elija el icono Add File (Añadir archivo)

    .
12. Vaya al directorio de trabajo en el que creó la clave privada y el certificado de firma de código, elija `tisigner.crt.der`, elija Open (Abrir) y, a continuación, elija Write (Escribir).

13. En la lista desplegable Action (Acción), elija Select MCU Image (Seleccionar imagen de MCU) y, a continuación, elija Browse (Explorar) para elegir la imagen de firmware que usará para escribir en su dispositivo (aws_demos.bin). Este archivo se encuentra en el directorio *freertos*/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug. Elija Abrir.

    a. En el cuadro de diálogo del archivo, confirme que el nombre de archivo es mcuflashimg.bin.

    b. Seleccione la casilla de verificación Vendor (Proveedor).

    c. En File Token (Token de archivo), escriba **1952007250**.

    d. En Private Key File Name (Nombre de archivo de clave privada), elija Browse (Explorar) y, a continuación, elija tisigner.key del directorio de trabajo donde creó el certificado de firma de código y la clave privada.

    e. En Certification File Name (Nombre del archivo de certificación), elija tisigner.crt.der.

    f. Elija Write (Escribir).

14. En el panel de navegación izquierdo, en Files (Archivos), elija Service Pack (Paquete de servicio).

15. En Service Pack File Name (Nombre del paquete de servicio), elija Browse (Explorar), vaya a simplelink_cc32x_sdk_*version*/tools/cc32xx_tools/servicepack-cc3x20, elija sp_3.7.0.1_2.0.0.0_2.2.0.6.bin y, a continuación, elija Open (Abrir).

16. En el panel izquierdo, en Files (Archivos), seleccione Trusted Root-Certificate Catalog (Catálogo de certificado raíz de confianza).

17. Desactive la casilla de verificación Use default Trusted Root-Certificate Catalog (Usar catálogo de certificado raíz de confianza predeterminado).

18. En Source File (Archivo de origen), elija Browse (Explorar), seleccione simplelink_cc32xx_sdk_***versión***/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst y, a continuación, elija Open (Abrir).

19. En Signature Source File (Archivo de origen de firma), elija Browse (Explorar), seleccione simplelink_cc32xx_sdk_***versión***/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst.signed_3220.bin y, a continuación, elija Open (Abrir).

20. Elija el botón

    para guardar el proyecto.

21. Elija el botón

.

22. Elija Program Image (Create and Program) (Programar imagen [Crear y programar]).
23. Una vez que el proceso de programación se haya completado, coloque el puente SOP en el primer conjunto de pines (posición = 0), restablezca la placa y vuelva a conectar su emulador de terminal a fin de asegurarse de que la salida es la misma que cuando depuró la demostración con Code Composer Studio. Anote el número de la versión de la aplicación de la salida de terminal. Utilizará este número de versión más tarde para verificar que el firmware se ha actualizado mediante una actualización OTA.

    El terminal debe mostrar una salida como la siguiente.

```
0 0 [Tmr Svc] Simple Link task created

Device came up in Station mode

1 369 [Tmr Svc] Starting key provisioning...
2 369 [Tmr Svc] Write root certificate...
3 467 [Tmr Svc] Write device private key...
4 568 [Tmr Svc] Write device certificate...
SL Disconnect...

5 664 [Tmr Svc] Key provisioning done...
Device came up in Station mode

Device disconnected from the AP on an ERROR..!!

[WLAN EVENT] STA Connected to the AP: Guest , BSSID: 11:22:a1:b2:c3:d4

[NETAPP EVENT] IP acquired by the device


Device has connected to Guest

Device IP Address is 111.222.3.44


6 1716 [OTA] OTA demo version 0.9.0
7 1717 [OTA] Creating MQTT Client...
8 1717 [OTA] Connecting to broker...
```

```
9 1717 [OTA] Sending command to MQTT task.
10 1717 [MQTT] Received message 10000 from queue.
11 2193 [MQTT] MQTT Connect was accepted. Connection established.
12 2193 [MQTT] Notifying task.
13 2194 [OTA] Command sent to MQTT task passed.
14 2194 [OTA] Connected to broker.
15 2196 [OTA Task] Sending command to MQTT task.
16 2196 [MQTT] Received message 20000 from queue.
17 2697 [MQTT] MQTT Subscribe was accepted. Subscribed.
18 2697 [MQTT] Notifying task.
19 2698 [OTA Task] Command sent to MQTT task passed.
20 2698 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/$next/
get/accepted

21 2699 [OTA Task] Sending command to MQTT task.
22 2699 [MQTT] Received message 30000 from queue.
23 2800 [MQTT] MQTT Subscribe was accepted. Subscribed.
24 2800 [MQTT] Notifying task.
25 2801 [OTA Task] Command sent to MQTT task passed.
26 2801 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/notify-
next

27 2814 [OTA Task] [OTA] Check For Update #0
28 2814 [OTA Task] Sending command to MQTT task.
29 2814 [MQTT] Received message 40000 from queue.
30 2916 [MQTT] MQTT Publish was successful.
31 2916 [MQTT] Notifying task.
32 2917 [OTA Task] Command sent to MQTT task passed.
33 2917 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:TI-LaunchPad ]
34 2917 [OTA Task] [OTA] Missing job parameter: execution
35 2917 [OTA Task] [OTA] Missing job parameter: jobId
36 2918 [OTA Task] [OTA] Missing job parameter: jobDocument
37 2918 [OTA Task] [OTA] Missing job parameter: ts_ota
38 2918 [OTA Task] [OTA] Missing job parameter: files
39 2918 [OTA Task] [OTA] Missing job parameter: streamname
40 2918 [OTA Task] [OTA] Missing job parameter: certfile
41 2918 [OTA Task] [OTA] Missing job parameter: filepath
42 2918 [OTA Task] [OTA] Missing job parameter: filesize
43 2919 [OTA Task] [OTA] Missing job parameter: sig-sha1-rsa
44 2919 [OTA Task] [OTA] Missing job parameter: fileid
45 2919 [OTA Task] [OTA] Missing job parameter: attr
47 3919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
48 4919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

```
49 5919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

Instalación de la versión inicial de firmware en ESP32 de Espressif

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Esta guía se ha escrito partiendo del supuesto de que ya ha realizado los pasos que se indican en Introducción a ESP32-DevKitC y ESP-WROVER-KIT de Espressif y Requisitos previos de las actualizaciones inalámbricas. Antes de intentar realizar una actualización OTA, es posible que desee ejecutar el proyecto de demostración MQTT descrito en Introducción a FreeRTOS para asegurarse de que su cadena de herramientas y placa están correctamente configuradas.

Para instalar una imagen de fábrica inicial en la placa

1. Abra *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, comente #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED y defina CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED o CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED.
2. Copie el certificado de firma de código con formato PEM SHA-256/ECDSA que ha generado en el Requisitos previos de actualización OTA en vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h. Debe tener el siguiente formato.

   ```
   #define otapalconfigCODE_SIGNING_CERTIFICATE  \
   "-----BEGIN CERTIFICATE-----\n" \
   "...base64 data...\n" \
   "-----END CERTIFICATE-----\n";
   ```
3. Con la actualización de demostración OTA seleccionada, siga los mismos pasos que se describen en Introducción a ESP32 para compilar e instalar la imagen. Si ha compilado e instalado el proyecto previamente, es posible que deba ejecutar `make clean` en primer lugar. Después de ejecutar `make flash monitor`, debería ver algo parecido a lo siguiente. El orden

de algunos mensajes puede variar, ya que la aplicación de demostración ejecuta varias tareas a la vez.

```
I (28) boot: ESP-IDF v3.1-dev-322-gf307f41-dirty 2nd stage bootloader
I (28) boot: compile time 16:32:33
I (29) boot: Enabling RNG early entropy source...
I (34) boot: SPI Speed : 40MHz
I (38) boot: SPI Mode : DIO
I (42) boot: SPI Flash Size : 4MB
I (46) boot: Partition Table:
I (50) boot: ## Label Usage Type ST Offset Length
I (57) boot: 0 nvs WiFi data 01 02 00010000 00006000
I (64) boot: 1 otadata OTA data 01 00 00016000 00002000
I (72) boot: 2 phy_init RF data 01 01 00018000 00001000
I (79) boot: 3 ota_0 OTA app 00 10 00020000 00100000
I (87) boot: 4 ota_1 OTA app 00 11 00120000 00100000
I (94) boot: 5 storage Unknown data 01 82 00220000 00010000
I (102) boot: End of partition table
I (106) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x14784
 ( 83844) map
I (144) esp_image: segment 1: paddr=0x000347ac vaddr=0x3ffb0000 size=0x023ec
 ( 9196) load
I (148) esp_image: segment 2: paddr=0x00036ba0 vaddr=0x40080000 size=0x00400
 ( 1024) load
I (151) esp_image: segment 3: paddr=0x00036fa8 vaddr=0x40080400 size=0x09068
 ( 36968) load
I (175) esp_image: segment 4: paddr=0x00040018 vaddr=0x400d0018 size=0x719b8
 (465336) map
I (337) esp_image: segment 5: paddr=0x000b19d8 vaddr=0x40089468 size=0x04934
 ( 18740) load
I (345) esp_image: segment 6: paddr=0x000b6314 vaddr=0x400c0000 size=0x00000 ( 0)
 load
I (353) boot: Loaded app from partition at offset 0x20000
I (353) boot: ota rollback check done
I (354) boot: Disabling RNG early entropy source...
I (360) cpu_start: Pro cpu up.
I (363) cpu_start: Single core mode
I (368) heap_init: Initializing. RAM available for dynamic allocation:
I (375) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (381) heap_init: At 3FFC0748 len 0001F8B8 (126 KiB): DRAM
I (387) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (393) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (400) heap_init: At 4008DD9C len 00012264 (72 KiB): IRAM
```

```
I (406) cpu_start: Pro cpu start user code
I (88) cpu_start: Starting scheduler on PRO CPU.
I (113) wifi: wifi firmware version: f79168c
I (113) wifi: config NVS flash: enabled
I (113) wifi: config nano formating: disabled
I (113) system_api: Base MAC address is not set, read default base MAC address from
 BLK0 of EFUSE
I (123) system_api: Base MAC address is not set, read default base MAC address from
 BLK0 of EFUSE
I (133) wifi: Init dynamic tx buffer num: 32
I (143) wifi: Init data frame dynamic rx buffer num: 32
I (143) wifi: Init management frame dynamic rx buffer num: 32
I (143) wifi: wifi driver task: 3ffc73ec, prio:23, stack:4096
I (153) wifi: Init static rx buffer num: 10
I (153) wifi: Init dynamic rx buffer num: 32
I (163) wifi: wifi power manager task: 0x3ffcc028 prio: 21 stack: 2560
0 6 [main] WiFi module initialized. Connecting to AP <Your_WiFi_SSID>...
I (233) phy: phy_version: 383.0, 79a622c, Jan 30 2018, 15:38:06, 0, 0
I (233) wifi: mode : sta (30:ae:a4:80:0a:04)
I (233) WIFI: SYSTEM_EVENT_STA_START
I (363) wifi: n:1 0, o:1 0, ap:255 255, sta:1 0, prof:1
I (1343) wifi: state: init -> auth (b0)
I (1343) wifi: state: auth -> assoc (0)
I (1353) wifi: state: assoc -> run (10)
I (1373) wifi: connected with <Your_WiFi_SSID>, channel 1
I (1373) WIFI: SYSTEM_EVENT_STA_CONNECTED
1 302 [IP-task] vDHCPProcess: offer c0a86c13ip
I (3123) event: sta ip: 192.168.108.19, mask: 255.255.224.0, gw: 192.168.96.1
I (3123) WIFI: SYSTEM_EVENT_STA_GOT_IP
2 302 [IP-task] vDHCPProcess: offer c0a86c13ip
3 303 [main] WiFi Connected to AP. Creating tasks which use network...
4 304 [OTA] OTA demo version 0.9.6
5 304 [OTA] Creating MQTT Client...
6 304 [OTA] Connecting to broker...
I (4353) wifi: pm start, type:0

I (8173) PKCS11: Initializing SPIFFS
I (8183) PKCS11: Partition size: total: 52961, used: 0
7 1277 [OTA] Connected to broker.
8 1280 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/$next/get/accepted
I (12963) ota_pal: prvPAL_GetPlatformImageState
I (12963) esp_ota_ops: [0] aflags/seq:0x2/0x1, pflags/seq:0xffffffff/0x0
```

```
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:<Your_Thing_Name> ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
[ ... ]
```

4. La placa ESP32 está ahora a la escucha de actualizaciones OTA. El comando `make flash monitor` lanza el monitor ESP-IDF. Pulse Ctrl+] para salir. También puede utilizar su programa favorito de terminal TTY (por ejemplo, PuTTY, Tera Term o GNU Screen) para escuchar salidas en serie de la placa. Tenga en cuenta que la conexión al puerto serie de la placa puede provocar un reinicio.

Instalación de la versión inicial de firmware en Nordic nRF52840 DK

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Esta guía se ha escrito partiendo del supuesto de que ya ha realizado los pasos que se indican en Introducción a Nordic nRF52840-DK yRequisitos previos de las actualizaciones inalámbricas. Antes de intentar realizar una actualización OTA, es posible que desee ejecutar el proyecto de demostración MQTT descrito en Introducción a FreeRTOS para asegurarse de que su cadena de herramientas y placa están correctamente configuradas.

Para instalar una imagen de fábrica inicial en la placa

1. Abrir *freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h.
2. Reemplace `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` por `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Con la actualización de demostración OTA seleccionada, siga los mismos pasos que se describen en Introducción a Nordic nRF52840-DK para compilar e instalar la imagen.

   Debería ver un resultado similar a este.

```
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/your-thing-name/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:your-thing-name ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

```
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

La placa está ahora a la escucha de actualizaciones OTA.

Firmware inicial en el simulador de Windows

Cuando se utiliza el simulador de Windows, no hay necesidad de instalar una versión inicial del firmware. El simulador de Windows forma parte de la aplicación `aws_demos`, que también incluye el firmware.

Instalación de la versión inicial de firmware en una placa personalizada

Con su IDE, compile el proyecto `aws_demos`, asegurándose de incluir la biblioteca OTA.
Para obtener más información sobre la estructura del código fuente de FreeRTOS, consulte Demostraciones de FreRTOS.

Asegúrese de incluir el certificado de firma de código, la clave privada y la cadena de confianza del certificado en el proyecto de FreeRTOS o en el dispositivo.

Con la herramienta adecuada, grabe la aplicación en la placa y asegúrese de que se ejecuta correctamente.

Actualización de la versión del firmware

El agente de OTA incluido con FreeRTOS comprueba la versión de cualquier actualización y solo la instala si es más reciente que la versión de firmware existente. Los pasos siguientes muestran cómo aumentar la versión de firmware de la aplicación de demostración OTA.

1. Abra el proyecto `aws_demos` en su IDE.
2. Localice el archivo **`/vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h`** e incremente el valor de APP_VERSION_BUILD.
3. Para programar una actualización en una plataforma Renesas rx65n con un tipo de archivo distinto de 0 (archivos que no son de firmware), debe firmar el archivo con la herramienta Renesas Secure Flash Programmer; de lo contrario, no pasará la verificación de firma en el dispositivo. La herramienta crea un paquete de archivos firmado con la extensión `.rsu`, que es un tipo de archivo propiedad de Renesas. La herramienta se puede encontrar en Github. Puede utilizar el siguiente comando de ejemplo para generar la imagen:

```
"Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure
 Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
```

4. Vuelva a compilar el proyecto.

Debe copiar su actualización de firmware en el bucket de Amazon S3 que ha creado, como se describe en Creación de un bucket de Amazon S3 para almacenar la actualización. El nombre del archivo que debe copiar a Amazon S3 depende de la plataforma de hardware que utilice:

- CC3220SF-LAUNCHXL de Texas Instruments: `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- ESP32 de Espressif: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

Creación de una actualización OTA (consola de AWS IoT)

1. En el panel de navegación de la consola de AWS IoT, elija Administrar y, a continuación, elija Acciones remotas y Trabajos.
2. Seleccione Create job (Crear trabajo).
3. En Tipo de trabajo, seleccione Crear trabajo de actualización OTA de FreeRTOS y, a continuación, elija Siguiente.
4. En Propiedades del trabajo, introduzca un nombre de trabajo y (de forma opcional) introduzca una descripción del trabajo y, a continuación, seleccione Siguiente.
5. Puede implementar una actualización OTA en un único dispositivo o en un grupo de dispositivos. En Dispositivos que se van a actualizar, seleccione uno o más objetos o grupos de objetos en el menú desplegable.
6. En Seleccionar el protocolo para la transferencia de archivos, seleccione HTTP o MQTT, o seleccione ambos para permitir que cada dispositivo determine el protocolo que debe utilizar.
7. En Firme y elija el archivo, seleccione Firmar un archivo nuevo por mí.
8. En Perfil de firma de código, elija Crear nuevo perfil.
9. En Create a code signing profile (Crear un perfil de firma de código), introduzca un nombre para el perfil de firma del código.

    a. En Plataforma de hardware del dispositivo, elija su plataforma de hardware.

> **Note**
>
> En esta lista solo se muestran las plataformas de hardware que han sido calificadas para FreeRTOS. Si va a probar un plataforma no cualificada y está utilizando el conjunto de cifrado ECDSA P-256 SHA-256 para firma, puede elegir el perfil de firma de código de Windows Simulator para crear una firma compatible. Si utiliza una plataforma no cualificada y un conjunto de cifrado distinto a ECDSA P-256 SHA-256 para la firma, puede utilizar la firma de código para AWS IoT o puede firmar la actualización de firmware usted mismo. Para obtener más información, consulte Firma digital de la actualización de firmware.

b. En Certificado de firma de código, seleccione Seleccionar un certificado existente y, a continuación, seleccione un certificado importado previamente o seleccione Importar un nuevo certificado de firma de código, seleccione sus archivos y, después, Importar para importar un certificado nuevo.

c. En Pathname of code signing certificate on device (Nombre de ruta del certificado de firma de código en el dispositivo), escriba el nombre de la ruta completo para el certificado de firma de código en el dispositivo. En la mayoría de los dispositivos, puede dejar este campo en blanco. Para el simulador de Windows y para los dispositivos que colocan el certificado en una ubicación de archivo específica, introduzca aquí el nombre de la ruta.

> **Important**
>
> En CC3220SF-LAUNCHXL de Texas Instruments, no incluya un carácter de barra inclinada (**/**) delante del nombre del archivo si el certificado de firma de código está en la raíz del sistema de archivos. De lo contrario, la actualización OTA falla durante la autenticación con un error `file not found`.

d. Seleccione Create (Crear).

10. En Archivo, seleccione Seleccionar un archivo existente y, a continuación, elija Examinar S3. Se muestra la lista de buckets de Amazon S3 disponibles. Elija el bucket que contiene la actualización de firmware y, a continuación, elija su actualización de firmware en el bucket.

> **Note**
>
> Los proyectos de demostración de Curiosity PIC32MZEF de Microchip producen dos imágenes binarias con nombres predeterminados de `mplab.production.bin` y `mplab.production.ota.bin`. Utilice el segundo archivo cuando carga una imagen para una actualización OTA.

11. En Ruta del archivo en el dispositivo, escriba el nombre de ruta completo en la ubicación de su dispositivo en la que el trabajo OTA copiará la imagen del firmware. Esta ubicación varía en función de la plataforma.

    > **Important**
    >
    > En CC3220SF-LAUNCHXL de Texas Instruments debido a restricciones de seguridad, el nombre de ruta de la imagen de firmware debe ser `/sys/mcuflashimg.bin`.

12. En Tipo de archivo, introduzca un valor entero comprendido entre 0 y 255. El tipo de archivo que introduzca se añadirá al documento de trabajo que se entrega a la MCU. El desarrollador del firmware/software de la MCU es el propietario total de lo que debe hacer con este valor. Los posibles escenarios incluyen una MCU que tenga un procesador secundario cuyo firmware se pueda actualizar de forma independiente del procesador principal. Cuando el dispositivo recibe un trabajo de actualización OTA, puede usar el tipo de archivo para identificar el procesador al que se destina la actualización.
13. En Rol de IAM, elija un rol de acuerdo con las instrucciones de Crear un rol de servicio de actualizaciones OTA.
14. Elija Siguiente.
15. Introduzca un ID y una descripción para su trabajo de actualización OTA.
16. En Tipo de trabajo, seleccione El trabajo se realizará después de implementarse en los dispositivos o grupos seleccionados (instantánea).
17. Seleccione la configuración opcional adecuada para su trabajo (Job executions rollout (Implementación de ejecuciones de trabajos), Job abort (Cancelación de trabajos), Job executions timeout (Tiempo de espera de ejecuciones de trabajos) y Tags (Etiquetas)).
18. Seleccione Crear.

Para usar una imagen de firmware previamente firmada

1. En Seleccionar y firmar la imagen de firmware, elija Seleccionar una imagen de firmware previamente firmada.
2. En Pathname of firmware image on device (Nombre de ruta de la imagen de firmware en el dispositivo), escriba el nombre de ruta completo en la ubicación de su dispositivo en la que el trabajo de OTA copiará la imagen del firmware. Esta ubicación varía en función de la plataforma.
3. En Trabajo de firma de código anterior, elija Seleccionar, a continuación, elija el trabajo de firma de código anterior para firmar la imagen de firmware que utiliza para la actualización OTA.

Uso de una imagen de firmware firmada personalizada

1. En Seleccionar y firmar la imagen de firmware, elija Usar mi imagen de firmware firmada personalizada.
2. En Pathname of code signing certificate on device (Nombre de ruta del certificado de firma de código en el dispositivo), escriba el nombre de la ruta completo para el certificado de firma de código en el dispositivo. En la mayoría de los dispositivos, puede dejar este campo en blanco. Para el simulador de Windows y para los dispositivos que colocan el certificado en una ubicación de archivo específica, introduzca aquí el nombre de la ruta.
3. En Pathname of firmware image on device (Nombre de ruta de la imagen de firmware en el dispositivo), escriba el nombre de ruta completo en la ubicación de su dispositivo en la que el trabajo de OTA copiará la imagen del firmware. Esta ubicación varía en función de la plataforma.
4. En Firma, pegue la firma en formato PEM.
5. En Original hash algorithm (Algoritmo hash original), elija el algoritmo hash que usó al crear la firma del archivo.
6. En Original encryption algorithm (Algoritmo de cifrado original), elija el algoritmo que usó al crear la firma del archivo.
7. En Seleccionar la imagen de firmware en Amazon S3, elija el bucket de Amazon S3 y la imagen de firmware firmada en el bucket de Amazon S3.

Una vez que haya especificado la información de firma de código, especifique el tipo de trabajo de actualización OTA, el rol de servicio y un ID para su actualización.

> **Note**
>
> No utilice información de identificación personal en el ID de trabajo de su actualización OTA. Los ejemplos de información de identificación personal incluyen:
>
> - Nombres.
> - Direcciones IP.
> - Direcciones de correo electrónico.
> - Ubicaciones.
> - Datos bancarios.
> - Información médica.

1. En Tipo de trabajo, seleccione El trabajo se realizará después de implementarse en los dispositivos o grupos seleccionados (instantánea).
2. En Rol de IAM para el trabajo de actualización de OTA, elija el rol de servicio de OTA.
3. Escriba un ID alfanumérico para el trabajo y seleccione Create (Crear).

El trabajo aparece en la consola de AWS IoT con el estado EN CURSO.

> **Note**
>
> - La consola de AWS IoT no actualiza el estado de los trabajos de forma automática. Actualice el navegador para ver las actualizaciones.

Conecte el terminal UART (serie) a su equipo. Debería ver una salida que indique que el dispositivo está descargando el firmware actualizado.

Una vez que el dispositivo descarga el firmware actualizado, se reinicia y, a continuación, instala el firmware. Puede ver de lo que sucede en el terminal UART.

Para ver un tutorial que muestra cómo utilizar la consola para crear una actualización OTA, consulte Aplicación de demostración de actualizaciones transparentes.

Creación de una actualización OTA con la AWS CLI

Cuando utiliza la AWS CLI para crear una actualización OTA:

1. Firmar digitalmente la imagen de firmware.
2. Crear una secuencia de su imagen de firmware firmada digitalmente.
3. Comenzar un trabajo de actualización OTA.

Firma digital de la actualización de firmware

Cuando utiliza la AWS CLI para realizar actualizaciones OTA, puede utilizar la firma de código para AWS IoT o firmar usted mismo la actualización de firmware. Para obtener una lista de los algoritmos de hash y firma criptográfica compatibles con la firma de código para AWS IoT, consulte SigningConfigurationOverrides. Si desea utilizar un algoritmo criptográfico que no sea compatible con la firma de código para AWS IoT, tiene que firmar los binarios de firmware antes de cargarlo en Amazon S3.

Firma de la imagen de firmware con la firma de código para AWS IoT

Para firmar la imagen de firmware mediante la firma de código para AWS IoT, puede utilizar uno de los SDK o herramientas de la línea de comandos de AWS. Para obtener más información sobre la firma de código para AWS IoT, consulte Firma de código para AWS IoT.

Después de instalar y configurar las herramientas de firma de código, copie su imagen de firmware sin firmar en el bucket de Amazon S3 e inicie un trabajo de firma de código con los siguientes comandos de la AWS CLI. El comando put-signing-profile crea un perfil reutilizable de firma de código. El comando start-signing-job inicia el trabajo de firma.

```
aws signer put-signing-profile \
    --profile-name your_profile_name \
    --signing-material certificateArn=arn:aws:acm::your-region:your-aws-account-
id:certificate/your-certificate-id \
    --platform your-hardware-platform \
    --signing-parameters certname=your_certificate_path_on_device
```

```
aws signer start-signing-job \
    --source
 's3={bucketName=your_s3_bucket,key=your_s3_object_key,version=your_s3_object_version_id}'
 \
    --destination 's3={bucketName=your_destination_bucket}' \
```

```
    --profile-name your_profile_name
```

> **Note**
>
> Los valores de *your-source-bucket-name* y *your-destination-bucket-name* pueden apuntar al mismo bucket de Amazon S3.

Esto son los parámetros de los comandos put-signing-profile y start-signing-job:

`source`

Especifica la ubicación del firmware sin firmar en un bucket de S3.

- `bucketName`: el nombre del bucket de S3.
- `key`: la clave (nombre de archivo) del firmware en su bucket de S3.
- `version`: la versión de S3 del firmware en su bucket de S3. Esto es diferente de la versión de firmware. Para encontrarla, vaya a la consola de Amazon S3, elija su bucket y, en la parte superior de la página, junto a Versiones, elija Mostrar.

`destination`

El destino del dispositivo en el que se copiará el firmware firmado en el bucket de S3. El formato de este parámetro es el mismo que el del parámetro `source`.

`signing-material`

El ARN del certificado de firma de código. Este ARN se genera al importar su certificado a ACM.

`signing-parameters`

Un mapa de pares de clave-valor para la firma. Puede incluir toda la información que desea utilizar durante la firma.

> **Note**
>
> Este parámetro es necesario cuando se crea un perfil de firma de código para firmar actualizaciones OTA con la firma de código para AWS IoT.

`platform`

El `platformId` de la plataforma de hardware en la que se va a distribuir la actualización OTA.

Para obtener una lista de las plataformas disponibles y sus valores de `platformId`, utilice el comando `aws signer list-signing-platforms`.

El trabajo de firma comienza y escribe la imagen de firmware firmada en el bucket de Amazon S3 de destino. El nombre del archivo de la imagen de firmware firmada es un GUID. Necesitará este nombre de archivo para crear una secuencia. Para encontrar el nombre del archivo, vaya a la consola de Amazon S3 y elija su bucket. Si no ve un archivo con un nombre de archivo GUID, actualice el navegador.

El comando muestra un ARN de trabajo y un ID de trabajo. Necesitará estos valores más adelante. Para obtener más información sobre la firma de código para AWS IoT, consulte Firma de código para AWS IoT.

Firma manual de la imagen de firmware

Firme digitalmente la imagen de firmware y cargue la imagen de firmware firmada en su bucket de Amazon S3.

Creación de una secuencia de actualización de firmware

Una secuencia es una interfaz abstracta a los datos que puede consumir un dispositivo. Una secuencia puede ocultar la complejidad de obtener acceso a los datos almacenados en diferentes ubicaciones o en diferentes servicios en la nube. El servicio de administrador de actualizaciones OTA le permite utilizar varios datos, almacenados en varias ubicaciones de Amazon S3, para realizar una actualización OTA.

Al crear una actualización OTA de AWS IoT, también puede crear una secuencia que contenga la actualización de firmware firmada. Cree un archivo JSON (`stream.json`) que identifique su imagen de firmware firmada. El archivo JSON debe contener lo siguiente.

```
[
  {
    "fileId":"your_file_id",
    "s3Location":{
      "bucket":"your_bucket_name",
      "key":"your_s3_object_key"
    }
  }
]
```

Estos son los atributos del archivo JSON:

`fileId`

Un número entero arbitrario entre 0 y 255 que identifica la imagen de firmware.

`s3Location`

El bucket y la clave para el firmware que se va a transmitir.

`bucket`

El bucket de Amazon S3 donde se almacena la imagen de firmware sin firmar.

`key`

El nombre del archivo de la imagen de firmware firmada en el bucket de Amazon S3. Puede encontrar este valor en la consola de Amazon S3. Para ello, debe consultar el contenido de su bucket.

Si utiliza Code Signing para AWS IoT, el nombre de archivo es un GUID generado por Code Signing para AWS IoT.

Utilice el comando create-stream de la AWS CLI para crear un flujo.

```
aws iot create-stream \
    --stream-id your_stream_id \
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

Estos son los argumentos del comando create-stream de la AWS CLI:

`stream-id`

Una cadena arbitraria para identificar la secuencia.

`description`

Una descripción opcional de la secuencia.

`files`

Una o varias referencias a archivos JSON que contienen datos sobre las imágenes de firmware que se van a transmitir. El archivo JSON contiene los siguientes atributos:

`fileId`

Un ID de archivo arbitrario.

`s3Location`

El nombre del bucket donde se almacena la imagen de firmware firmada y la clave (nombre del archivo) de la imagen de firmware firmada.

`bucket`

El bucket de Amazon S3 donde se almacena la imagen de firmware firmada.

`key`

La clave (nombre del archivo) de la imagen de firmware firmada.

Cuando se utiliza Code Signing para AWS IoT, esta clave es un GUID.

A continuación se muestra un ejemplo de un archivo `stream.json`.

```
[
    {
        "fileId":123,
        "s3Location": {
            "bucket":"codesign-ota-bucket",
            "key":"48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
        }
    }
]
```

`role-arn`

El rol de servicio OTA que también concede acceso al bucket de Amazon S3 donde se almacena la imagen del firmware.

Para encontrar la clave del objeto de Amazon S3 de la imagen de firmware firmada, utilice el comando aws signer describe-signing-job --job-id ***`my-job-id`***, donde `my-job-id` es el ID de trabajo que muestra el comando create-signing-job de la AWS CLI. La salida del comando describe-signing-job contiene la clave de la imagen de firmware firmada.

```
... text deleted for brevity ...
  "signedObject": {
```

```
    "s3": {
      "bucketName": "ota-bucket",
      "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
  }
... text deleted for brevity ...
```

Creación de una actualización OTA

Utilice el comando create-ota-update de la AWS CLI para crear un trabajo de actualización OTA.

> **Note**
>
> No utilice información de identificación personal (PII) en el ID de trabajo de su actualización OTA. Los ejemplos de información de identificación personal incluyen:
>
> - Nombres.
> - Direcciones IP.
> - Direcciones de correo electrónico.
> - Ubicaciones.
> - Datos bancarios.
> - Información médica.

```
aws iot  create-ota-update \
    --ota-update-id value \
    [--description value] \
    --targets value \
    [--protocols value] \
    [--target-selection value] \
    [--aws-job-executions-rollout-config value] \
    [--aws-job-presigned-url-config value] \
    [--aws-job-abort-config value] \
    [--aws-job-timeout-config value] \
    --files value \
    --role-arn value \
    [--additional-parameters value] \
    [--tags value]  \
    [--cli-input-json value] \
    [--generate-cli-skeleton]
```

Formato `cli-input-json`

```
{
  "otaUpdateId": "string",
  "description": "string",
  "targets": [
    "string"
  ],
  "protocols": [
    "string"
  ],
  "targetSelection": "string",
  "awsJobExecutionsRolloutConfig": {
    "maximumPerMinute": "integer",
    "exponentialRate": {
      "baseRatePerMinute": "integer",
      "incrementFactor": "double",
      "rateIncreaseCriteria": {
        "numberOfNotifiedThings": "integer",
        "numberOfSucceededThings": "integer"
      }
    }
  },
  "awsJobPresignedUrlConfig": {
    "expiresInSec": "long"
  },
  "awsJobAbortConfig": {
    "abortCriteriaList": [
      {
        "failureType": "string",
        "action": "string",
        "thresholdPercentage": "double",
        "minNumberOfExecutedThings": "integer"
      }
    ]
  },
  "awsJobTimeoutConfig": {
    "inProgressTimeoutInMinutes": "long"
  },
  "files": [
    {
      "fileName": "string",
      "fileType": "integer",
      "fileVersion": "string",
```

```
      "fileLocation": {
        "stream": {
          "streamId": "string",
          "fileId": "integer"
        },
        "s3Location": {
          "bucket": "string",
          "key": "string",
          "version": "string"
        }
      },
      "codeSigning": {
        "awsSignerJobId": "string",
        "startSigningJobParameter": {
          "signingProfileParameter": {
            "certificateArn": "string",
            "platform": "string",
            "certificatePathOnDevice": "string"
          },
          "signingProfileName": "string",
          "destination": {
            "s3Destination": {
              "bucket": "string",
              "prefix": "string"
            }
          }
        },
        "customCodeSigning": {
          "signature": {
            "inlineDocument": "blob"
          },
          "certificateChain": {
            "certificateName": "string",
            "inlineDocument": "string"
          },
          "hashAlgorithm": "string",
          "signatureAlgorithm": "string"
        }
      },
      "attributes": {
        "string": "string"
      }
    }
  ],
```

```
  "roleArn": "string",
  "additionalParameters": {
    "string": "string"
  },
  "tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

Campos **`cli-input-json`**

Nombre	Tipo	Descripción
`otaUpdateId`	string (máximo: 128 mínimo:1)	El ID de la actualización OTA que se va a crear.
`description`	string (máximo: 2028)	La descripción de la actualización OTA.
`targets`	Lista	Los dispositivos que van a recibir actualizaciones OTA.
`protocols`	Lista	El protocolo utilizado para transferir la imagen de actualización de OTA. Los valores válidos son [HTTP], [MQTT], [HTTP, MQTT]. Cuando se especifican HTTP y MQTT, el dispositivo de destino puede elegir el protocolo.
`targetSelection`	string	Especifica si la actualización seguirá ejecutándose (CONTINUOUS) o si se completará después de que

Nombre	Tipo	Descripción
		todos los objetos especific ados como destino hayan completado la actualización (SNAPSHOT). Si el estado es CONTINUOUS, es posible también que la actualización solo pueda ejecutarse en un objeto cuando se detecte un cambio en un destino. Por ejemplo, se ejecutará una actualización en un objeto cuando este se añada a un grupo de destino, incluso después de que los objetos originales del grupo completen la actualización. Valores válidos: CONTINUOUS | SNAPSHOT. Enum: CONTINUOUS | SNAPSHOT
`awsJobExecutionsRo lloutConfig`		Configuración de la implement ación de las actualizaciones OTA.
`maximumPerMinute`	entero (máximo: 1000 mínimo:1)	El número máximo de ejecuciones de trabajos de actualización de OTA iniciadas por minuto.

Nombre	Tipo	Descripción
`exponentialRate`		La tasa de aumento para el despliegue de un trabajo. Este parámetro le permite definir un aumento de tasa exponenci al para el despliegue de un trabajo.
`baseRatePerMinute`	entero (máximo: 1000 mínimo:1)	El número mínimo de objetos del que se notificará de un trabajo pendiente, por minuto, al empezar a desplegar el trabajo. Esta es la tasa inicial del despliegue.
`rateIncreaseCriteria`		Los criterios para iniciar el aumento en la tasa de despliegue de un trabajo. AWS IoT admite hasta un dígito después del decimal (por ejemplo, 1.5, pero no 1,55).
`numberOfNotifiedTh ings`	entero (mínimo:1)	Cuando se haya notificado a esta cantidad de cosas, se iniciará un aumento en la tasa de despliegue.
`numberOfSucceededT hings`	entero (mínimo:1)	Cuando esta cantidad de cosas hayan tenido éxito en la ejecución de su trabajo, se iniciará un aumento en la tasa de despliegue.
`awsJobPresignedUrl Config`		Información de configuración de las URL prefirmadas.

Nombre	Tipo	Descripción
`expiresInSec`	long	Tiempo (en segundos) durante el que las URL prefirmad as son válidas. Los valores válidos oscilan entre 60 y 3600. El valor predeterm inado es 1800 segundos. Las URL prefirmadas se generan cuando se recibe una solicitud para el documento de trabajo.
`awsJobAbortConfig`		Criterios que determinan cuándo y cómo se produce la detención de un trabajo.
`abortCriteriaList`	Lista	Lista de criterios que determinan cuándo y cómo detener el trabajo.
`failureType`	string	Tipo de errores de ejecución de trabajos que pueden iniciar una detención del trabajo. enum: FAILED | REJECTED | TIMED_OUT | ALL
`action`	string	Tipo de acción de trabajo que se va a realizar para iniciar la detención del trabajo. enum: CANCEL
`minNumberOfExecute dThings`	entero (mínimo:1)	El número mínimo de objetos que deben recibir notificac iones de ejecución de trabajos antes de que el trabajo se pueda detener.

Nombre	Tipo	Descripción
`awsJobTimeoutConfig`		Especifica la cantidad de tiempo que cada dispositivo tiene para finalizar su ejecución del trabajo. Un temporizador se pone en marcha cuando el estado de ejecución del trabajo se establece en `IN_PROGRESS` . Si el estado de ejecución del trabajo no se establece en otro estado terminal antes de que el temporizador agota el plazo, se establecerá automáticamente en `TIMED_OUT` .

Nombre	Tipo	Descripción
`inProgressTimeoutInMinutes`	long	Especifica la cantidad de tiempo, en minutos, que tiene este dispositivo para finalizar la ejecución de este trabajo. El intervalo de tiempo de espera puede estar en cualquier momento entre 1 minuto y 7 días (1 a 10 080 minutos). El temporizador en curso no se puede actualizar y se aplicará a todas las ejecuciones de trabajo para el trabajo. Cada vez que un trabajo permanece en el estado de ejecución IN_PROGRESS durante un periodo superior a este intervalo, la ejecución del trabajo producirá un error y cambiará al estado terminal `TIMED_OUT` .
`files`	Lista	Los archivos que se transmiten mediante la actualización OTA.
`fileName`	string	El nombre del archivo.
`fileType`	entero Rango máx.: 255; mín.: 0	Un valor entero que puede incluir en el documento de trabajo para que sus dispositivos puedan identificar el tipo de archivo recibido de la nube.
`fileVersion`	string	La versión del archivo.

Nombre	Tipo	Descripción
`fileLocation`		La ubicación del firmware actualizado.
`stream`		La secuencia que contiene la actualización OTA.
`streamId`	string (máximo: 128 mínimo:1)	El ID de transmisión.
`fileId`	entero (máximo:255 mínimo:0)	El ID de un archivo asociado con un flujo.
`s3Location`		La ubicación del firmware actualizado en S3.
`bucket`	string (mínimo:1)	El bucket de S3.
`key`	string (mínimo:1)	La clave de S3.
`version`	string	La versión del bucket de S3.
`codeSigning`		El método de firma de código del archivo.
`awsSignerJobId`	string	El ID de AWSSignerJob que se creó para firmar el archivo.
`startSigningJobParameter`		Describe el trabajo de firma de código.
`signingProfileParameter`		Describe el perfil de firma de código.

Nombre	Tipo	Descripción
certificateArn	string	ARN del certificado.
platform	string	La plataforma de hardware de su dispositivo.
certificatePathOnDevice	string	La ubicación del certificado de firma de código en el dispositivo.
signingProfileName	string	El nombre del perfil de firma de código.
destination		La ubicación para escribir el archivo cuyo código se ha firmado.
s3Destination		Describe la ubicación del firmware actualizado en S3.
bucket	string (mínimo:1)	El bucket de S3 que contiene el firmware actualizado.
prefix	string	El prefijo S3.
customCodeSigning		Un método personalizado para la firma de código de un archivo.
signature		La firma del archivo.
inlineDocument	blob	Una representación binaria con cifrado base64 de la firma usada para firmar el código.
certificateChain		La cadena de certificados.
certificateName	string	El nombre del certificado.

Nombre	Tipo	Descripción
`inlineDocument`	string	Una representación binaria con cifrado base64 de la cadena de certificados de firma de código.
`hashAlgorithm`	string	El algoritmo hash que se utiliza para la firma de código del archivo.
`signatureAlgorithm`	string	El algoritmo de firma que se utiliza para la firma de código del archivo.
`attributes`	map	Una lista de pares nombre-atributo.
`roleArn`	string (máximo: 2048 mínimo:20)	El rol de IAM que concede a AWS IoT acceso a Amazon S3, a los trabajos de AWS IoT y a los recursos de firma de código de AWS para crear un trabajo de actualización OTA.
`additionalParameters`	map	Una lista de parámetros de actualizaciones OTA adicionales que son pares nombre-valor.
`tags`	Lista	Metadatos que se pueden utilizar para administrar actualizaciones.
`Key`	string (máximo: 128 mínimo:1)	La clave de la etiqueta.

Nombre	Tipo	Descripción
`Value`	string (máximo: 256 mínimo:1)	El valor de la etiqueta.

Output

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

Campos de salida de la AWS CLI

Nombre	Tipo	Descripción
`otaUpdateId`	string (máximo: 128 mínimo:1)	El ID de la actualización OTA.
`awsIotJobId`	string	El ID de trabajo de AWS IoT asociado con la actualización OTA.
`otaUpdateArn`	string	El ARN de actualización OTA.
`awsIotJobArn`	string	El ARN de trabajo de AWS IoT asociado con la actualización OTA.
`otaUpdateStatus`	string	El estado de la actualización OTA. Enum: CREATE_PENDING | CREATE_IN_PROGRESS

Nombre	Tipo	Descripción
		| CREATE_COMPLETE | CREATE_FAILED

A continuación, se muestra un ejemplo de un archivo JSON que se pasa en el comando create-ota-update que utiliza la firma de código para AWS IoT.

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

A continuación, se muestra un ejemplo de un archivo JSON que se transfiere al comando create-ota-update de la AWS CLI que utiliza un archivo insertado para proporcionar material de firma de código personalizada.

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId": 123
      }
    },
    "codeSigning": {
      "customCodeSigning":{
        "signature":{
```

```
          "inlineDocument":"your_signature"
        },
        "certificateChain": {
          "certificateName": "your_certificate_name",
          "inlineDocument":"your_certificate_chain"
        },
        "hashAlgorithm":"your_hash_algorithm",
        "signatureAlgorithm":"your_signature_algorithm"
      }
    }
  }
]
```

A continuación, se muestra un ejemplo de un archivo JSON transferido al comando create-ota-update de la AWS CLIque permite a la OTA de FreeRTOS comenzar un trabajo de firma de código y crear una secuencia y un perfil de firma del código.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "myTestProfile",
        "signingProfileParameter": {
          "certificateArn": "your_certificate_arn",
          "platform": "your_platform_id",
          "certificatePathOnDevice": "certificate_path"
        },
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
```

```
    }
  }
]
```

A continuación, se muestra un ejemplo de un archivo JSON transferido al comando create-ota-update de la AWS CLI y que crea una actualización OTA que comienza un trabajo de firma de código con un perfil existente y utiliza la secuencia especificada.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_s3_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "your_unique_profile_name",
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
    }
  }
]
```

A continuación, se muestra un ejemplo de un archivo JSON transferido al comando create-ota-update de la AWS CLI que permite a la OTA de FreeRTOS crear una secuencia con un ID de trabajo de firma de código existente.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
```

```
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

A continuación, se muestra un ejemplo de un archivo JSON transferido al comando create-ota-update de la AWS CLI que crea una actualización OTA. La actualización crea una secuencia desde el objeto de S3 especificado y utiliza firma de código personalizada.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "customCodeSigning": {
        "signature":{
          "inlineDocument":"your_signature"
        },
        "certificateChain": {
          "inlineDocument":"your_certificate_chain",
          "certificateName": "your_certificate_path_on_device"
        },
        "hashAlgorithm":"your_hash_algorithm",
        "signatureAlgorithm":"your_sig_algorithm"
      }
    }
  }
]
```

Listado de actualizaciones OTA

Puede utilizar el comando list-ota-updates de la AWS CLI para obtener una lista de todas las actualizaciones OTA.

```
aws iot list-ota-updates
```

La salida del comando list-ota-updates es como se muestra a continuación.

```
{
  "otaUpdates": [

    {
      "otaUpdateId": "my_ota_update2",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update2",
      "creationDate": 1522778769.042
    },
    {
      "otaUpdateId": "my_ota_update1",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update1",
      "creationDate": 1522775938.956
    },
    {
      "otaUpdateId": "my_ota_update",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update",
      "creationDate": 1522775151.031
    }
  ]
}
```

Obtener información sobre una actualización OTA

Puede utilizar el comando get-ota-update de la AWS CLI para obtener el estado de creación o eliminación de una actualización OTA.

```
aws iot get-ota-update --ota-update-id your-ota-update-id
```

El resultado del comando get-ota-update tendrá un aspecto similar al siguiente.

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "ota-update-001",
        "otaUpdateArn": "arn:aws:iot:region:123456789012:otaupdate/ota-update-001",
        "creationDate": 1575414146.286,
        "lastModifiedDate": 1575414149.091,
        "targets": [
            "arn:aws:iot:region:123456789012:thing/myDevice"
```

```
        ],
        "protocols": [ "HTTP" ],
        "awsJobExecutionsRolloutConfig": {
            "maximumPerMinute": 0
        },
        "awsJobPresignedUrlConfig": {
            "expiresInSec": 1800
        },
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
                "fileName": "my_firmware.bin",
                "fileType": 1,
                "fileLocation": {
                    "s3Location": {
                        "bucket": "my-bucket",
                        "key": "my_firmware.bin",
                        "version": "AvP3bfJC9gyqnwoxPHuTqM5GWENt4iii"
                    }
                },
                "codeSigning": {
                    "awsSignerJobId": "b7a55a54-fae5-4d3a-b589-97ed103737c2",
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "my-profile-name",
                        "destination": {
                            "s3Destination": {
                                "bucket": "some-ota-bucket",
                                "prefix": "SignedImages/"
                            }
                        }
                    },
                    "customCodeSigning": {}
                }
            }
        ],
        "otaUpdateStatus": "CREATE_COMPLETE",
        "awsIotJobId": "AFR_OTA-ota-update-001",
        "awsIotJobArn": "arn:aws:iot:region:123456789012:job/AFR_OTA-ota-update-001"
    }
}
```

Los valores devueltos para `otaUpdateStatus` incluyen lo siguiente:

CREATE_PENDING

La creación de una actualización OTA está pendiente.

CREATE_IN_PROGRESS

Se está creando una actualización OTA.

CREATE_COMPLETE

Se ha creado una actualización OTA.

CREATE_FAILED

La creación de una actualización OTA ha fracasado.

DELETE_IN_PROGRESS

Se está eliminando una actualización OTA.

DELETE_FAILED

La eliminación de una actualización OTA ha fracasado.

> **Note**
>
> Para obtener el estado de ejecución de una actualización OTA después de que se haya creado, debe utilizar el comando describe-job-execution. Para obtener más información, consulte Descripción de una ejecución de trabajo.

Eliminación de datos relacionados con OTA

En estos momentos, no puede usar la consola de AWS IoT para eliminar secuencias o actualizaciones OTA. Puede utilizar la AWS CLI para eliminar secuencias, actualizaciones OTA y trabajos de AWS IoT creados durante una actualización OTA.

Eliminar una secuencia OTA

Cuando crea una actualización OTA que utiliza MQTT, puede utilizar la línea de comandos o la consola de AWS IoT para crear una secuencia que divida el firmware en fragmentos para que se pueda enviar a través de MQTT. Puede eliminar esta secuencia con el comando delete-stream de la AWS CLI, como se muestra en el siguiente ejemplo.

```
aws iot delete-stream --stream-id your_stream_id
```

Eliminar una actualización OTA

Al crear una actualización OTA, se crean los siguientes elementos:

- Una entrada en la base de datos del trabajo de actualización OTA.
- Un trabajo de AWS IoT para llevar a cabo la actualización.
- Una ejecución de trabajo de AWS IoT para cada dispositivo que se está actualizando.

El comando delete-ota-update elimina la entrada en la base de datos del trabajo de actualización OTA únicamente. Debe utilizar el comando delete-job para eliminar el trabajo de AWS IoT.

Utilice el comando delete-ota-update para eliminar una actualización OTA.

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

`ota-update-id`

El ID de la actualización OTA que se va a eliminar.

`delete-stream`

Elimina la secuencia asociada a la actualización OTA.

`force-delete-aws-job`

Elimina el trabajo de AWS IoT asociado a la actualización OTA. Si no se establece esta marca y el trabajo se encuentra en estado `In_Progress`, el trabajo no se elimina.

Eliminar un trabajo de IoT creado para una actualización OTA

FreeRTOS crea un trabajo de AWS IoT al crear una actualización OTA. También se crea una ejecución de trabajo para cada dispositivo que procesa el trabajo. Puede utilizar el comando delete-job de la AWS CLI para eliminar un trabajo y sus ejecuciones de trabajo asociadas.

```
aws iot delete-job --job-id your-job-id --no-force
```

El parámetro `no-force` especifica que solo se pueden eliminar los trabajos que están en estado terminal (COMPLETED o CANCELLED). Puede eliminar un trabajo que se encuentra en un estado no terminal pasando el parámetro `force`. Para obtener más información, consulte API DeleteJob.

> **Note**
>
> Eliminar un trabajo con un estado IN_PROGRESS interrumpe todas las ejecuciones de trabajo que están IN_PROGRESS en sus dispositivos y puede dar lugar a que un dispositivo quede en un estado no determinista. Asegúrese de que todos los dispositivos que ejecutan un trabajo que se ha eliminado pueden recuperarse a un estado conocido.

Según el número de ejecuciones de trabajo creadas para el trabajo y otros factores, se podría tardar unos minutos en borrar un trabajo. Mientras el trabajo se está eliminando, su estado es DELETION_IN_PROGRESS. Al intentar eliminar o cancelar un trabajo cuyo estado ya es DELETION_IN_PROGRESS, se producirá un error.

Puede utilizar delete-job-execution para eliminar una ejecución de trabajo. Es posible que desee eliminar una ejecución de trabajo cuando un pequeño número de dispositivos no sea capaz de procesar un trabajo. Este elimina la ejecución de trabajo para un único dispositivo, como se muestra en el siguiente ejemplo.

```
aws iot delete-job-execution --job-id your-job-id --thing-name
                    your-thing-name --execution-number your-job-execution-number --no-
force
```

Al igual que con el comando delete-job de la AWS CLI, puede transferir el parámetro `--force` a delete-job-execution para forzar la eliminación de una ejecución de trabajo. Para obtener más información, consulte API DeleteJobExecution.

> **Note**
>
> Eliminar una ejecución de trabajo con un estado IN_PROGRESS interrumpe todas las ejecuciones de trabajo que están IN_PROGRESS en sus dispositivos y puede dar lugar a que un dispositivo quede en un estado no determinista. Asegúrese de que todos los dispositivos que ejecutan un trabajo que se ha eliminado pueden recuperarse a un estado conocido.

Para obtener más información sobre el uso de la aplicación de demostración de actualizaciones OTA, consulte Aplicación de demostración de actualizaciones transparentes.

Servicio OTA Update Manager

El servicio Over-the-Air (OTA) Update Manager proporciona una forma de:

- Crear una actualización OTA y los recursos que utiliza, incluidos un trabajo de AWS IoT, una secuencia de AWS IoT y la firma de código.
- Obtener información sobre la actualización OTA.
- Enumerar todas las actualizaciones OTA asociadas a su cuenta de AWS.
- Elimina una actualización OTA.

Una actualización OTA es una estructura de datos mantenidos por el servicio OTA Update Manager. Contiene:

- Un ID de actualización OTA.
- Una descripción opcional de la actualización OTA.
- Una lista de dispositivos para actualizar (destinos).
- El tipo de actualización OTA: CONTINUA o INSTANTÁNEA. Consulte la sección Trabajos de la Guía para desarrolladores de AWS IoT para obtener información sobre el tipo de actualización que necesita.
- Protocolo utilizado para realizar la actualización OTA: [MQTT], [HTTP] o [MQTT, HTTP]. Cuando se especifica MQTT y HTTP, la configuración del dispositivo determina el protocolo utilizado.
- Una lista de archivos para enviar a los dispositivos de destino.
- El rol de IAM que concede a AWS IoT acceso a Amazon S3, a los trabajos de AWS IoT y a los recursos de firma de código de AWS para crear un trabajo de actualización OTA.
- Una lista opcional de pares de nombre-valor definidos por el usuario.

Las actualizaciones OTA están pensadas para actualizar firmware de dispositivos, pero puede utilizarlas para enviar archivos a uno o más dispositivos registrados en AWS IoT. Cuando envíe actualizaciones de firmware de manera inalámbrica, le recomendamos que las firme digitalmente para que los dispositivos que las reciben puedan verificar que no se han manipulado durante el trayecto.

Puede enviar imágenes de firmware actualizadas utilizando el protocolo HTTP o MQTT, dependiendo de la configuración que elija. Puede firmar las actualizaciones de firmware con la Firma de código para FreeRTOS o puede usar sus propias herramientas de firma de código.

Para disponer de mayor control sobre el proceso, puede utilizar la API CreateStream para crear una secuencia al enviar actualizaciones a través de MQTT. En algunos casos, puede modificar el código del agente Freertos para ajustar el tamaño de los bloques que envía y recibe.

Cuando se crea una actualización OTA, el servicio OTA Manager genera un trabajo de AWS IoT para notificar a los dispositivos que hay una actualización disponible. El Agente de OTA de FreeRTOS se ejecuta en sus dispositivos y escucha los mensajes de actualización. Cuando una actualización está disponible, solicita la imagen de actualización del firmware a través de HTTP o MQTT y almacena los archivos localmente. Comprueba la firma digital de los archivos descargados y si es válida, instala la actualización de firmware. Si no va a utilizar FreeRTOS, debe implementar su propio Agente de OTA para escuchar y descargar actualizaciones y realizar las operaciones de instalación.

Integración del Agente de OTA en la aplicación

El Agente de actualización inalámbrica (OTA) se ha diseñado para simplificar la cantidad de código que debe escribir para añadir la funcionalidad de actualización OTA a su producto. Esta carga de integración consiste principalmente en inicializar el Agente de OTA y crear una función de devolución de llamada personalizada para responder a los mensajes de eventos de agente de OTA. Durante la inicialización, el sistema operativo, MQTT, HTTP (si se utiliza HTTP para la descarga de archivos) y las interfaces de implementación específica de la plataforma (PAL) se transfieren al agente OTA. Los búferes también se pueden inicializar y transferir al agente OTA.

> **Note**
>
> Aunque la integración de la característica de actualización OTA en su aplicación es bastante sencilla, no es suficiente estar familiarizado con la integración de código en dispositivos sino que se requieren conocimientos más extensos. Para familiarizarse con cómo configurar su cuenta de AWS con objetos de AWS IoT, credenciales, certificados de firma de código, dispositivos de aprovisionamiento y trabajos de actualización OTA, consulte la sección Requisitos previos de FreeRTOS.

Administración de conexiones

El Agente de OTA utiliza el protocolo MQTT para todas las operaciones de comunicación de control que implican servicios de AWS IoT, pero no administra la conexión MQTT. Para asegurarse de que el Agente de OTA no interfiere con la política de administración de la conexión de la aplicación, la conexión MQTT, que incluye funcionalidades para desconectarse y volver a conectarse, debe ser administrada por la aplicación del usuario principal. El archivo se puede descargar a través del protocolo MQTT o HTTP. Puede elegir el protocolo al crear el trabajo de OTA. Si elige MQTT, el Agente de OTA utiliza la misma conexión para las operaciones de control y para la descarga de archivos.

Demostración de OTA sencilla

A continuación, se muestra un fragmento de una demostración de OTA sencilla que muestra cómo el Agente se conecta al agente de MQTT e inicializa el Agente de OTA. En este ejemplo, configuramos la demostración para que utilice la devolución de llamada de la aplicación OTA predeterminada y devuelva algunas estadísticas una vez por segundo. Por cuestiones de brevedad, omitimos algunos detalles de esta demostración.

La demostración de OTA también muestra la gestión de conexiones MQTT mediante la supervisión de la devolución de llamadas de desconexión y el restablecimiento de la conexión. Cuando se produce una desconexión, la demostración suspende primero las operaciones del agente OTA y, a continuación, intenta restablecer la conexión MQTT. Los intentos de reconexión de MQTT se retrasan un tiempo, que se incrementa exponencialmente hasta alcanzar un valor máximo, a lo que se añade una fluctuación. Si se restablece la conexión, el agente OTA continúa sus operaciones.

Para ver un ejemplo práctico que utiliza el agente MQTT de AWS IoT, consulte el código de demostración de OTA del directorio `demos/ota`.

Dado que el Agente de OTA es su propia tarea, el retraso intencional de un segundo en este ejemplo solo afecta a esta aplicación. No tiene ningún impacto en el rendimiento del Agente.

```
static BaseType_t prvRunOTADemo( void )
{
    /* Status indicating a successful demo or not. */
    BaseType_t xStatus = pdFAIL;

    /* OTA library return status. */
    OtaErr_t xOtaError = OtaErrUninitialized;

    /* OTA event message used for sending event to OTA Agent.*/
```

```
    OtaEventMsg_t xEventMsg = { 0 };

    /* OTA interface context required for library interface functions.*/
    OtaInterfaces_t xOtaInterfaces;

    /* OTA library packet statistics per job.*/
    OtaAgentStatistics_t xOtaStatistics = { 0 };

    /* OTA Agent state returned from calling OTA_GetState.*/
    OtaState_t xOtaState = OtaAgentStateStopped;

    /* Set OTA Library interfaces.*/
    prvSetOtaInterfaces( &xOtaInterfaces );

    /************************** Init OTA Library. **************************/

    if( ( xOtaError = OTA_Init( &xOtaBuffer,
                                &xOtaInterfaces,
                                ( const uint8_t * ) ( democonfigCLIENT_IDENTIFIER ),
                                prvOtaAppCallback ) ) != OtaErrNone )
    {
        LogError( ( "Failed to initialize OTA Agent, exiting = %u.",
                    xOtaError ) );
    }
    else
    {
        xStatus = pdPASS;
    }

    /************************ Create OTA Agent Task. ************************/

    if( xStatus == pdPASS )
    {
        xStatus = xTaskCreate( prvOTAAgentTask,
                               "OTA Agent Task",
                               otaexampleAGENT_TASK_STACK_SIZE,
                               NULL,
                               otaexampleAGENT_TASK_PRIORITY,
                               NULL );

        if( xStatus != pdPASS )
        {
            LogError( ( "Failed to create OTA agent task:" ) );
        }
```

```
    }

    /****************************** Start OTA ******************************/

    if( xStatus == pdPASS )
    {
        /* Send start event to OTA Agent.*/
        xEventMsg.eventId = OtaAgentEventStart;
        OTA_SignalEvent( &xEventMsg );
    }

    /******************** Loop and display OTA statistics ********************/

    if( xStatus == pdPASS )
    {
        while( ( xOtaState = OTA_GetState() ) != OtaAgentStateStopped )
        {
            /* Get OTA statistics for currently executing job. */
            if( xOtaState != OtaAgentStateSuspended )
            {
                OTA_GetStatistics( &xOtaStatistics );

                LogInfo( ( " Received: %u   Queued: %u   Processed: %u   Dropped: %u",
                           xOtaStatistics.otaPacketsReceived,
                           xOtaStatistics.otaPacketsQueued,
                           xOtaStatistics.otaPacketsProcessed,
                           xOtaStatistics.otaPacketsDropped ) );
            }

            vTaskDelay( pdMS_TO_TICKS( otaexampleEXAMPLE_TASK_DELAY_MS ) );
        }
    }

    return xStatus;
}
```

A continuación, se muestra el flujo general de esta aplicación de demostración:

- Cree un contexto de Agente de MQTT.
- Conéctese a su punto de enlace de AWS IoT.
- Inicialice el Agente de OTA.
- Bucle que permite un trabajo de actualización OTA y genera estadísticas una vez por segundo.

- Si el MQTT se desconecta, suspende las operaciones del agente OTA.
- Intenta conectarse de nuevo con un retardo y una fluctuación exponenciales.
- Si se vuelve a conectar, reanuda las operaciones del agente OTA.
- Si el agente se detiene, espera un segundo e intenta volver a conectarse.

Uso de la devolución de llamada de la aplicación para los eventos del agente OTA

En el ejemplo anterior se utilizó `prvOtaAppCallback` como controlador de devolución de llamadas para los eventos de agente OTA. (Consulte el cuarto parámetro de la llamada a la API `OTA_Init`). Si desea implementar una gestión personalizada de los eventos de finalización, debe cambiar la gestión predeterminada en la demostración/aplicación OTA. Durante el proceso de OTA, el Agente OTA puede enviar una de las siguientes enumeraciones de eventos al controlador de devolución de llamada. Es responsabilidad del desarrollador de la aplicación decidir cómo y cuándo gestionar estos eventos.

```
/**
 * @ingroup ota_enum_types
 * @brief OTA Job callback events.
 *
 * After an OTA update image is received and authenticated, the agent calls the user
 * callback (set with the @ref OTA_Init API) with the value OtaJobEventActivate to
 * signal that the device must be rebooted to activate the new image. When the device
 * boots, if the OTA job status is in self test mode, the agent calls the user callback
 * with the value OtaJobEventStartTest, signaling that any additional self tests
 * should be performed.
 *
 * If the OTA receive fails for any reason, the agent calls the user callback with
 * the value OtaJobEventFail instead to allow the user to log the failure and take
 * any action deemed appropriate by the user code.
 *
 * See the OtaImageState_t type for more information.
 */
typedef enum OtaJobEvent
{
    OtaJobEventActivate = 0,       /*!< @brief OTA receive is authenticated and ready
 to activate. */
    OtaJobEventFail = 1,           /*!< @brief OTA receive failed. Unable to use this
 update. */
    OtaJobEventStartTest = 2,      /*!< @brief OTA job is now in self test, perform
 user tests. */
```

```
    OtaJobEventProcessed = 3,      /*!< @brief OTA event queued by OTA_SignalEvent is
 processed. */
    OtaJobEventSelfTestFailed = 4, /*!< @brief OTA self-test failed for current job. */
    OtaJobEventParseCustomJob = 5, /*!< @brief OTA event for parsing custom job
 document. */
    OtaJobEventReceivedJob = 6,    /*!< @brief OTA event when a new valid AFT-OTA job
 is received. */
    OtaJobEventUpdateComplete = 7, /*!< @brief OTA event when the update is completed.
 */
    OtaLastJobEvent = OtaJobEventStartTest
} OtaJobEvent_t;
```

El Agente de OTA puede recibir una actualización en segundo plano durante el procesamiento activo de la aplicación principal. El objetivo de la entrega de estos eventos es permitir a la aplicación que decida si se puede realizar la acción inmediatamente o si debe aplazarse hasta después de la finalización de otro procesamiento específico de la aplicación. De este modo, evita una interrupción no prevista de su dispositivo durante el procesamiento activo (por ejemplo, limpieza) que se debería a un restablecimiento después de una actualización de firmware. Estos son eventos de trabajo recibidos por el controlador de devolución de llamada:

`OtaJobEventActivate`

Cuando el controlador de devolución de llamada recibe este evento, puede restablecer el dispositivo inmediatamente o programar una llamada para restablecer el dispositivo más tarde. Esto le permite posponer el restablecimiento del dispositivo y la fase de autodiagnóstico, si es necesario.

`OtaJobEventFail`

Cuando el controlador de devolución de llamada recibe este evento, la actualización ha fallado. En este caso, no tiene que hacer nada. Es posible que desee generar un mensaje de registro o hacer algo específico de la aplicación.

`OtaJobEventStartTest`

La fase de autocomprobación está diseñada para permitir que el firmware recién actualizado se ejecute y realice una autocomprobación antes de determinar que funciona correctamente y confirmarlo en la última imagen de la aplicación permanente. Cuando se recibe y autentica una nueva actualización y el dispositivo se restablece, el Agente de OTA envía el evento `OtaJobEventStartTest` a la función de devolución de llamada cuando esté listo para realizar pruebas. El desarrollador puede añadir cualquier prueba que considere necesaria para

determinar si el firmware del dispositivo funciona correctamente después de la actualización. Cuando las pruebas de autodiagnóstico determinan que el firmware del dispositivo es de confianza, el código debe confirmar que el firmware es la nueva imagen permanente llamando a la función `OTA_SetImageState( OtaImageStateAccepted )`.

`OtaJobEventProcessed`

Se procesa el evento de OTA que `OTA_SignalEvent` ha puesto en cola, por lo que se pueden realizar operaciones de limpieza, como liberar los búferes de OTA.

`OtaJobEventSelfTestFailed`

La autocomprobación de OTA ha fallado para el trabajo actual. El procedimiento predeterminado para este evento consiste en cerrar el agente OTA y reiniciarlo para que el dispositivo vuelva a la imagen anterior.

`OtaJobEventUpdateComplete`

El evento de notificación de la finalización de la actualización del trabajo OTA.

Seguridad de OTA

Los siguientes son tres aspectos de la seguridad inalámbrica (OTA):

Seguridad de la conexión

El servicio OTA Update Manager se basa en los mecanismos de seguridad existentes, como la autenticación mutua de Transport Layer Security (TLS), que utiliza AWS IoT. El tráfico de actualización OTA pasa a través de la puerta de enlace del dispositivo de AWS IoT y utiliza mecanismos de seguridad de AWS IoT. Cada mensaje HTTP o MQTT de entrada y salida a través de la gateway del dispositivo se somete a estricta autenticación y autorización.

Autenticidad e integridad de las actualizaciones OTA

Es posible firmar el firmware digitalmente antes de una actualización OTA para asegurarse de que procede de una fuente de confianza y no se ha manipulado.

El servicio de administrador de actualizaciones OTA de FreeRTOS usa la firma de código para AWS IoT para firmar el firmware automáticamente. Para obtener más información, consulte Firma de código para AWS IoT.

El Agente de OTA, que se ejecuta en sus dispositivos, realiza comprobaciones de integridad en el firmware cuando llega al dispositivo.

Seguridad del operador

Cada llamada a la API que se realiza a través de la API de plano de control se somete a autenticación y autorización de firma de IAM estándar, versión 4. Para crear una implementación, debe tener permisos para invocar las API `CreateDeployment`, `CreateJob` y `CreateStream`. Además, en su política de bucket de Amazon S3 o ACL, debe conceder permisos de lectura a la entidad principal del servicio de AWS IoT para que sea posible acceder a la actualización de firmware almacenada en Amazon S3 durante el streaming.

Code Signing para AWS IoT

La consola de AWS IoT utiliza la Firma de código para AWS IoT para firmar automáticamente la imagen de firmware de cualquier dispositivo admitido por AWS IoT.

La firma de código para AWS IoT utiliza un certificado y una clave privada que importa a ACM. Puede utilizar un certificado autofirmado para realizar pruebas, pero recomendamos que obtenga un certificado de una autoridad de certificación (CA) comercial consolidada..

Los certificados de firma de código utilizan las extensiones `Key Usage` y `Extended Key Usage` de la versión 3 de X.509. La extensión `Key Usage` se establece en `Digital Signature` y la extensión `Extended Key Usage` se establece en `Code Signing`. Para obtener más información acerca de la firma de la imagen de código, consulte la Guía para desarrolladores acerca de la firma de código para AWS IoT y la Referencia de la API acerca de la firma de código para AWS IoT.

> **Note**
>
> Puede descargar el SDK de la firma de código para AWS IoT desde Herramientas para Amazon Web Services.

Solución de problemas de OTA

Las siguientes secciones contienen información para ayudarle a solucionar problemas con actualizaciones OTA.

Temas

- Configuración de registros de Cloudwatch para actualizaciones OTA
- Registrar llamadas a la API de OTA de AWS IoT con AWS CloudTrail
- Obtención de los detalles del error de CreateOTaupdate mediante la AWS CLI

- Obtener códigos de error de OTA con AWS CLI
- Solución de problemas con las actualizaciones OTA en varios dispositivos
- Solucionar problemas de actualizaciones OTA con CC3220SF Launchpad de Texas Instruments

Configuración de registros de Cloudwatch para actualizaciones OTA

El servicio de actualización OTA admite registros con Amazon CloudWatch. Puede utilizar la consola de AWS IoT para habilitar y configurar los Registros de Amazon CloudWatch para actualizaciones OTA. Para obtener más información, consulte Cloudwatch Logs.

Para habilitar el registro, debe crear un rol de IAM y configurar el registro de actualización OTA.

> **Note**
>
> Antes de habilitar el registro de actualización OTA, asegúrese de comprender bien los permisos de acceso a los registros de CloudWatch. Los usuarios con acceso a CloudWatch Logs podrán consultar la información de depuración. Para obtener más información, consulte Autenticación y control de acceso de Amazon CloudWatch Logs.

Crear un rol de registro y habilitar el registro

Utilice la consola de AWS IoT para crear un rol de registro y habilitar el registro.

1. En el panel de navegación, seleccione Configuración.
2. En Registros, elija Editar.
3. En Nivel de detalle, elija Depuración.
4. En Definir rol, elija Crear nuevo para crear un rol de IAM para el registro.
5. En Nombre, escriba un nombre único para su rol. El rol se creará con todos los permisos necesarios.
6. Elija Actualizar.

Registros de actualización OTA

El servicio de actualización OTA publica registros en su cuenta cuando se produce alguna de las siguientes situaciones:

- Se crea una actualización OTA.

- Se completa una actualización OTA.
- Se crea un trabajo de firma de código.
- Se completa un trabajo de firma de código.
- Se crea un trabajo de AWS IoT.
- Se completa un trabajo de AWS IoT.
- Se crea una secuencia.

Puede ver los registros en la consola de CloudWatch.

Para ver una actualización de OTA en los registros de CloudWatch

1. En el panel de navegación, elija Logs (Registros).
2. En Grupos de registros, elija AWSIoTLogsV2.

Los registros de actualización OTA pueden contener las siguientes propiedades:

accountId

El ID de cuenta de AWS en la que en que se generó el registro.

actionType

La acción que generó el registro. Esta propiedad se puede establecer en uno de los siguientes valores:

- `CreateOTAUpdate`: se creó una actualización OTA.
- `DeleteOTAUpdate`: se eliminó una actualización OTA.
- `StartCodeSigning`: se inició un trabajo de firma de código.
- `CreateAWSJob`: se creó un trabajo de AWS IoT.
- `CreateStream`: se creó una secuencia.
- `GetStream`: se envió una solicitud de transmisión a la función de entrega de archivos basada en MQTT de AWS IoT.
- `DescribeStream`: se envió una solicitud de información sobre una transmisión a la función de entrega de archivos basada en MQTT de AWS IoT.

awsJobId

El ID de trabajo de AWS IoT que generó el registro.

clientId

El ID de cliente de MQTT que ha realizado la solicitud que generó el registro.

clientToken

El token de cliente asociado con la solicitud que generó el registro.

details

Más información acerca de la operación que generó el registro.

logLevel

El nivel de registro del registro. En el caso de los registros de actualización OTA, este siempre está establecido en `DEBUG`.

otaUpdateId

El ID de actualización OTA que generó el registro.

protocolo

El protocolo usado para realizar la solicitud que generó el registro.

status

El estado de la operación que generó el registro. Los valores válidos son:

- Correcto
- Error

streamId

El ID de secuencia de AWS IoT que generó el registro.

timestamp

La hora en que se generó el registro.

topicName

Un tema de MQTT usado para realizar la solicitud que generó el registro.

Registros de ejemplo

A continuación, se muestra un registro de ejemplo generado cuando se inicia un trabajo de firma de código:

```
{
    "timestamp": "2018-07-23 22:59:44.955",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "StartCodeSigning",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Start code signing job. The request status is SUCCESS."
}
```

A continuación, se muestra un registro de ejemplo generado cuando se crea un trabajo de AWS IoT:

```
{
    "timestamp": "2018-07-23 22:59:45.363",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateAWSJob",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "awsJobId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Create AWS Job The request status is SUCCESS."
}
```

A continuación, se muestra un registro de ejemplo generado cuando se crea una actualización OTA:

```
{
    "timestamp": "2018-07-23 22:59:45.413",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateOTAUpdate",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "OTAUpdate creation complete. The request status is SUCCESS."
}
```

A continuación, se muestra un registro de ejemplo generado cuando se crea una secuencia:

```
{
    "timestamp": "2018-07-23 23:00:26.391",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
```

```
    "status": "Success",
    "actionType": "CreateStream",
    "otaUpdateId": "3d3dc5f7-3d6d-47ac-9252-45821ac7cfb0",
    "streamId": "6be2303d-3637-48f0-ace9-0b87b1b9a824",
    "details": "Create stream. The request status is SUCCESS."
}
```

A continuación, se muestra un registro de ejemplo generado cuando se elimina una actualización OTA:

```
{
    "timestamp": "2018-07-23 23:03:09.505",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DeleteOTAUpdate",
    "otaUpdateId": "9bdd78fb-f113-4001-9675-1b595982292f",
    "details": "Delete OTA Update. The request status is SUCCESS."
}
```

A continuación, se muestra un registro de ejemplo generado cuando un dispositivo solicita una secuencia de la función de entrega de archivos basada en MQTT:

```
{
    "timestamp": "2018-07-25 22:09:02.678",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "GetStream",
    "protocol": "MQTT",
    "clientId": "b9d2e49c-94fe-4ed1-9b07-286afed7e4c8",
    "topicName": "$aws/things/b9d2e49c-94fe-4ed1-9b07-286afed7e4c8/
streams/1e51e9a8-9a4c-4c50-b005-d38452a956af/get/json",
    "streamId": "1e51e9a8-9a4c-4c50-b005-d38452a956af",
    "details": "The request status is SUCCESS."
}
```

A continuación, se muestra un registro de ejemplo generado cuando un dispositivo llama a la API `DescribeStream`:

```
{
```

```
    "timestamp": "2018-07-25 22:10:12.690",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DescribeStream",
    "protocol": "MQTT",
    "clientId": "581075e0-4639-48ee-8b94-2cf304168e43",
    "topicName": "$aws/things/581075e0-4639-48ee-8b94-2cf304168e43/streams/71c101a8-
bcc5-4929-9fe2-af563af0c139/describe/json",
    "streamId": "71c101a8-bcc5-4929-9fe2-af563af0c139",
    "clientToken": "clientToken",
    "details": "The request status is SUCCESS."
}
```

Registrar llamadas a la API de OTA de AWS IoT con AWS CloudTrail

FreeRTOS viene integrado en CloudTrail, un servicio que captura las llamadas a la API de OTA de AWS IoT y envía los archivos de registro al bucket de Amazon S3 que especifique. CloudTrail captura las llamadas a la API de su código a las API de OTA de AWS IoT. Con la información que recopila CloudTrail, puede determinar la solicitud a la OTA de AWS IoT que se realizó, la dirección IP de origen desde la que se realizó la solicitud, quién realizó la solicitud, cuándo se realizó, etcétera.

Para obtener más información acerca de CloudTrail, incluido cómo configurarlo y habilitarlo, consulte la Guía de usuario de AWS CloudTrail.

Información de FreeRTOS en CloudTrail

Cuando el registro de CloudTrail está habilitado en su cuenta de AWS, las llamadas a la API realizadas a las acciones de OTA de AWS IoT se rastrean en los archivos de registro de CloudTrail, donde se escriben con otros registros de servicios de AWS. CloudTrail determina cuándo debe crearse un nuevo archivo y escribir en él en función del periodo de tiempo y del tamaño del archivo.

CloudTrail registra las siguientes acciones de plano de control de OTA de AWS IoT:

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream

- CreateOTAUpdate
- GetOTAUpdate
- ListOTAUpdates
- DeleteOTAUpdate

> **Note**
>
> CloudTrail no registra las acciones del plano de datos (lado del dispositivo) de OTA de AWS IoT. Utilice CloudWatch para supervisarlas.

Cada entrada de registro contiene información sobre quién generó la solicitud. La información de identidad del usuario en la entrada de registro le ayuda a determinar lo siguiente:

- Si la solicitud se realizó con las credenciales raíz o del usuario de IAM de .
- 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 Elemento userIdentity de CloudTrail. Las acciones de OTA de AWS IoT se documentan en la Referencia de la API de OTA de AWS IoT.

Puede almacenar sus archivos de registro en su bucket de Amazon S3 durante todo el tiempo que desee, pero también puede definir reglas de ciclo de vida de Amazon S3 para archivar o eliminar archivos de registro automáticamente. De forma predeterminada, los archivos de registro se cifran con el cifrado del servidor (SSE) de Amazon S3.

Si desea recibir notificaciones cuando se entreguen los archivos de registro, puede configurar CloudTrail para que publique las notificaciones de Amazon SNS. Para obtener más información, consulte Configuración de notificaciones de Amazon SNS para CloudTrail.

También puede agregar archivos de registro de OTA de AWS IoT desde varias regiones de AWS y varias cuentas de AWS en un solo bucket de Amazon S3.

Para obtener más información, consulte Recepción de archivos de registro de CloudTrail de varias regiones y Recepción de archivos de registro de CloudTrail de varias cuentas.

Descripción de las entradas de archivos de registro de FreeRTOS

Los archivos de registro de CloudTrail pueden contener una o más entradas de registro. Cada entrada muestra varios eventos con formato JSON. Una entrada de registro representa una única solicitud de cualquier origen e incluye información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etcétera. Las entradas de registro no son un rastro de la pila ordenada de las llamadas API públicas, por lo que no aparecen en ningún orden específico.

En el ejemplo siguiente, se muestra una entrada de registro de CloudTrail para ilustrar el registro de una llamada a la acción `CreateOTAUpdate`.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:iam::your_aws_account:user/your_user_id",
        "accountId": "your_aws_account",
        "accessKeyId": "your_access_key_id",
        "userName": "your_username",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-23T17:27:08Z"
            }
        },
        "invokedBy": "apigateway.amazonaws.com"
    },
    "eventTime": "2018-08-23T17:27:19Z",
    "eventSource": "iot.amazonaws.com",
    "eventName": "CreateOTAUpdate",
    "awsRegion": "your_aws_region",
    "sourceIPAddress": "apigateway.amazonaws.com",
    "userAgent": "apigateway.amazonaws.com",
    "requestParameters": {
        "targets": [
            "arn:aws:iot:your_aws_region:your_aws_account:thing/Thing_CMH"
        ],
        "roleArn": "arn:aws:iam::your_aws_account:role/Role_FreeRTOSJob",
        "files": [
            {
                "fileName": "/sys/mcuflashimg.bin",
                "fileSource": {
```

```
                    "fileId": 0,
                    "streamId": "your_stream_id"
                },
                "codeSigning": {
                    "awsSignerJobId": "your_signer_job_id"
                }
            }
        ],
        "targetSelection": "SNAPSHOT",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "responseElements": {
        "otaUpdateArn": "arn:aws:iot:your_aws_region:your_aws_account:otaupdate/
FreeRTOSJob_CMH-23-1535045232806-92",
        "otaUpdateStatus": "CREATE_PENDING",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "requestID": "c9649630-a6f9-11e8-8f9c-e1cf2d0c9d8e",
    "eventID": "ce9bf4d9-5770-4cee-acf4-0e5649b845c0",
    "eventType": "AwsApiCall",
    "recipientAccountId": "recipient_aws_account"
}
```

Obtención de los detalles del error de CreateOTaupdate mediante la AWS CLI

Si se produce un error al crear un trabajo de actualización OTA, puede realizar algunas acciones para solucionar el problema. Cuando crea un trabajo de actualización OTA, el servicio de administrador de OTA crea un trabajo de IoT y lo programa para los dispositivos de destino, y este proceso también crea o usa otros tipos de recursos de AWS en su cuenta (un trabajo de firma de código, una transmisión de AWS IoT o un objeto de Amazon S3). Cualquier error que se produzca puede provocar que el proceso falle sin crear un trabajo de AWS IoT. En esta sección de solución de problemas, damos instrucciones sobre cómo recuperar los detalles del error.

1. Instalar y configurar la AWS CLI.
2. Ejecute `aws configure` e introduzca la siguiente información.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

Para obtener más información, consulte Configuración rápida con `aws configure`.

3. Ejecute:

```
aws iot get-ota-update --ota-update-id ota_update_job_001
```

Donde *ota_update_job_001* es el ID que asignó a la actualización de OTA cuando la creó.

4. La salida tendrá este aspecto:

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "ota_update_job_001",
        "otaUpdateArn":
 "arn:aws:iot:region:account_id:otaupdate/ota_update_job_001",
        "creationDate": 1584646864.534,
        "lastModifiedDate": 1584646865.913,
        "targets": [
            "arn:aws:iot:region:account_id:thing/thing_001"
        ],
        "protocols": [
            "MQTT"
        ],
        "awsJobExecutionsRolloutConfig": {},
        "awsJobPresignedUrlConfig": {},
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
               "fileName": "/12ds",
                "fileLocation": {
                    "s3Location": {
                        "bucket": "bucket_name",
                        "key": "demo.bin",
                        "version": "Z7X.TWSAS7JSi4rybc02nMdcE41W1tV3"
                    }
                },
                "codeSigning": {
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "signing_profile_name",
                        "destination": {
                            "s3Destination": {
                                "bucket": "bucket_name",
```

```
                                "prefix": "SignedImages/"
                            }
                        }
                    },
                    "customCodeSigning": {}
                }
            }
        ],
        "otaUpdateStatus": "CREATE_FAILED",
        "errorInfo": {
            "code": "AccessDeniedException",
            "message": "S3 object demo.bin not accessible. Please check
 your permissions (Service: AWSSigner; Status Code: 403; Error Code:
 AccessDeniedException; Request ID: 01d8e7a1-8c7c-4d85-9fd7-dcde975fdd2d)"
        }
    }
}
```

Si la creación ha fallado, el campo `otaUpdateStatus` de la salida del comando contendrá `CREATE_FAILED` y el campo `errorInfo`contendrá los detalles del error.

Obtener códigos de error de OTA con AWS CLI

1. Instalar y configurar la AWS CLI.
2. Ejecute `aws configure` e introduzca la siguiente información.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

Para obtener más información, consulte Configuración rápida con `aws configure`.

3. Ejecute:

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

Donde *JobID* es la cadena completa de ID del trabajo cuyo estado queremos obtener (se asoció al trabajo de actualización OTA cuando se creó) y *ThingName* es el nombre del objeto de AWS IoT con el que el dispositivo se ha registrado en AWS IoT

4. La salida tendrá este aspecto:

```
{
    "execution": {
        "jobId": "AFR_OTA-****************",
        "status": "FAILED",
        "statusDetails": {
            "detailsMap": {
                "reason": "0xEEEEEEEE: 0xffffffff"
            }
        },
        "thingArn": "arn:aws:iot:Region:AccountID:thing/ThingName",
        "queuedAt": 1569519049.9,
        "startedAt": 1569519052.226,
        "lastUpdatedAt": 1569519052.226,
        "executionNumber": 1,
        "versionNumber": 2
    }
}
```

En este ejemplo de resultado, el valor "`reason`" de "`detailsmap`" contiene dos campos: el campo que aparece como "0xEEEEEEEE" contiene el código de error genérico del agente de OTA y el campo que aparece como "0xffffffff" contiene el subcódigo. Los códigos de error genéricos aparecen en https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws__ota__agent_8h.html. Consulte los códigos de error con el prefijo "`kOTA_Err_`". El subcódigo puede ser un código específico de la plataforma o facilitar más detalles acerca del error genérico.

Solución de problemas con las actualizaciones OTA en varios dispositivos

Para realizar actualizaciones OTA en varios dispositivos (objetos) utilizando la misma imagen de firmware, implemente una función (por ejemplo, `getThingName()`) que recupere `clientcredentialIOT_THING_NAME` de la memoria no volátil. Asegúrese de que esta función lee el nombre del objeto en una parte de la memoria no volátil que no se sobrescriba con la actualización OTA y de que el nombre del objeto se aprovisione antes de ejecutar el primer trabajo. Si está

utilizando el flujo JITP, puede obtener el nombre del objeto a partir del nombre común del certificado del dispositivo.

Solucionar problemas de actualizaciones OTA con CC3220SF Launchpad de Texas Instruments

La plataforma CC3220SF Launchpad ofrece un mecanismo de detección de manipulaciones de software. Utiliza un contador de alertas de seguridad que aumenta cuando se produce una infracción de integridad. El dispositivo se bloquea cuando el contador de alertas de seguridad alcanza un umbral predeterminado (el valor predeterminado es 15) y el host recibe el evento asíncrono `SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT`. El dispositivo bloqueado tiene accesibilidad limitada. Para recuperar el dispositivo, puede reprogramarlo o utilizar el proceso de restauración a los valores de fábrica para volver a la imagen de fábrica. Para programar el comportamiento deseado, actualice el controlador de eventos asíncronos en `network_if.c`.

Bibliotecas FreeRTOS

Las bibliotecas de FreeRTOS proporcionan funcionalidad adicional al kernel de FreeRTOS y a sus bibliotecas internas. Puede usar las bibliotecas de FreeRTOS para redes y seguridad en aplicaciones integradas. Las bibliotecas de FreeRTOS también permiten que sus aplicaciones interactúen con los servicios de AWS IoT. FreeRTOS incluye bibliotecas que le permiten:

- Conectar dispositivos de manera segura a la nube de AWS IoT mediante MQTT y sombras de dispositivos.
- Descubrir y conectarse a núcleos de AWS IoT Greengrass.
- Administrar conexiones wifi.
- Escuchar y procesar Actualizaciones vía inalámbrica de FreeRTOS.

El directorio `libraries` contiene el código fuente de las bibliotecas de FreeRTOS. Existen funciones auxiliares que contribuyen a la implementación de la funcionalidad de la biblioteca. No se recomienda que cambie estas funciones auxiliares.

Bibliotecas de portabilidad de FreeRTOS

Las siguientes bibliotecas de portabilidad se incluyen en configuraciones de FreeRTOS que están disponibles para su descarga en la consola de FreeRTOS. Estas bibliotecas dependen de la plataforma. Su contenido cambia de acuerdo con su plataforma de hardware. Para obtener más información acerca de la portabilidad estas bibliotecas a un dispositivo, consulte la Guía de portabilidad de FreeRTOS.

Bibliotecas de portabilidad de FreeRTOS

Library	Referencia de la API	Descripción
Bluetooth de bajo consumo	Referencia de la API de Bluetooth Low Energy	Con la biblioteca de Bluetooth de bajo consumo de FreeRTOS, el microcontrolador puede comunicar se con el agente de MQTT de AWS IoT a través de un dispositivo de puerta de enlace. Para obtener más información, consulte Biblioteca de Bluetooth de bajo consumo.
Actualizaciones inalámbricas	Referencia de la API de actualiza ción inalámbrica de AWS IoT	La biblioteca de actualizaciones vía inalámbrica (OTA) de AWS IoT de FreeRTOS le permite administrar las notificaciones de actualización, descargar actualizaciones y realizar una verificación criptográfica de las actualizaciones de firmware en su dispositivo FreeRTOS. Para obtener más información, consulte Biblioteca de actualiza ciones inalámbricas de AWS IoT.
FreeRTOS+POSIX	Referencia de la API de FreeRTOS +POSIX	Puede utilizar la biblioteca FreeRTOS+POSIX para realizar la portabilidad de aplicacio nes compatibles con POSIX al ecosistema de FreeRTOS. Para obtener más información, consulte FreeRTOS+POSIX.
Sockets seguros	Referencia de la API de sockets seguros	Para obtener más información, consulte Biblioteca de sockets seguros.

Library	Referencia de la API	Descripción
FreeRTOS+TCP	Referencia de la API de FreeRTOS +TCP	FreeRTOS+TCP es una pila TCP/IP segura para subprocesos de código abierto escalable para FreeRTOS. Para obtener más información, consulte FreeRTOS+TCP.
Wifi	Referencia de la API Wi-Fi	La biblioteca Wi-Fi de FreeRTOS le permite comunicarse con la pila inalámbrica de nivel inferior del microcontrolador. Para obtener más información, consulte Biblioteca wifi.
corePKCS11		La biblioteca corePKCS11 es una implementación de referenci a del Estándar de criptografía de clave pública 11 para respaldar el aprovisionamiento y la autenticación de cliente de TLS. Para obtener más información, consulte Biblioteca corePKCS11.
TLS		Para obtener más información, consulte Transport Layer Security.
E/S común	Referencia común de la API de E/S	Para obtener más información, consulte E/S común.
Interfaz móvil	Referencia de la API de interfaz móvil	La biblioteca de interfaces móviles expone las capacidades de algunos módems móviles populares a través de una API uniforme. Para obtener más información, consulte Bibliotec a de interfaces móviles.

Bibliotecas de aplicaciones de FreeRTOS

Puede incluir las siguientes bibliotecas de aplicaciones independientes en la configuración de FreeRTOS para interactuar con los servicios de AWS IoT en la nube.

> **Note**
>
> Algunas de las bibliotecas de aplicaciones tienen las mismas API que las bibliotecas en el SDK de dispositivos de AWS IoT para Embedded C. Para estas bibliotecas, consulte la Referencia de la API C del SDK de dispositivos AWS IoT. Para obtener más información sobre el SDK de dispositivos de AWS IoT para C integrado, consulte SDK de dispositivos de AWS IoT para Embedded C.

Bibliotecas de aplicaciones de FreeRTOS

Library	Referencia de la API	Descripción
AWS IoT Device Defender	Referencia de la API del SDK C de Device Defender	La biblioteca de AWS IoT Device Defender de FreeRTOS conecta su dispositivo FreeRTOS a AWS IoT Device Defender. Para obtener más información, consulte Biblioteca de AWS IoT Device Defender.
AWS IoT Greengrass	Referencia de la API de Greengrass	La biblioteca de AWS IoT Greengrass de FreeRTOS conecta su dispositivo FreeRTOS a AWS IoT Greengrass. Para obtener más información, consulte Biblioteca de detección de AWS IoT Greengrass.
MQTT	Referencia de la API de la biblioteca de MQTT (v1.x.x)	La biblioteca coreMQTT proporcio na un cliente para su dispositivo FreeRTOS para publicar y suscribir

Library	Referencia de la API	Descripción
	Referencia de la API del agente de MQTT (v1) Referencia de la API del SDK C de MQTT (v2.x.x)	se a temas de MQTT. MQTT es el protocolo que utilizan los dispositivos para interactuar con AWS IoT. Para obtener más información acerca de la versión 3.0.0 de la biblioteca coreMQTT, consulte Biblioteca coreMQTT.
Agente coreMQTT	Referencia de la API de la biblioteca de agentes coreMQTT	La biblioteca de agentes coreMQTT es una API de alto nivel que añade seguridad de subprocesos a la biblioteca coreMQTT. Permite crear una tarea de agente MQTT dedicada que gestiona una conexión MQTT en segundo plano y no necesita la intervención de otras tareas. La biblioteca proporciona equivalentes seguros para subprocesos para las API de coreMQTT, por lo que se puede utilizar en entornos con varios subprocesos. Para obtener más información acerca de la biblioteca de agentes coreMQTT, consulte Biblioteca de agente coreMQTT.

Library	Referencia de la API	Descripción
Sombra de dispositivos AWS IoT	Referencia de la API del SDK C de sombras de dispositivos	La biblioteca de sombras de dispositivos de AWS IoT permite que su dispositivo FreeRTOS interactúe con sombras de dispositivos de AWS IoT. Para obtener más información, consulte Biblioteca de sombras de dispositivos de AWS IoT.

Configuración de bibliotecas de FreeRTOS

Los ajustes de configuración de FreeRTOS y el SDK de dispositivos de AWS IoT para C integrado se definen como constantes de preprocesador C. Establezca las opciones de configuración con un archivo de configuración global o mediante una opción de compilador como `-D` en `gcc`. Debido a que las opciones de configuración se definen como constantes de tiempo de compilación, una biblioteca debe recompilarse si se cambia una opción de configuración.

Si desea utilizar un archivo de configuración global para definir las opciones de configuración, cree y guarde el archivo con el nombre `iot_config.h` y, a continuación, colóquelo en su ruta de inclusión. En el archivo, utilice directivas `#define` para configurar las bibliotecas, demostraciones y pruebas de FreeRTOS.

Para obtener más información acerca de las opciones de configuración globales admitidas, consulte la Referencia de archivos de configuración global.

Biblioteca backoffAlgorithm

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

La biblioteca backoffAlgorithm es una biblioteca de utilidades que se utiliza para espaciar las retransmisiones repetidas del mismo bloque de datos, a fin de evitar la congestión de la red. Esta biblioteca calcula el período de espera para reintentar las operaciones de red (como una conexión de red fallida con el servidor) mediante un algoritmo de retroceso exponencial con fluctuación.

El retroceso exponencial con fluctuación se suele utilizar al reintentar una conexión o solicitud de red fallida a un servidor causada por una congestión de la red o por una carga excesiva en el servidor. Se utiliza para distribuir el tiempo de las solicitudes de reintento creadas por varios dispositivos que intentan conectarse a la red al mismo tiempo. En un entorno con una conectividad deficiente, un cliente puede desconectarse en cualquier momento, por lo que una estrategia de espera también ayuda al cliente a ahorrar batería al no intentar volver a conectarse repetidamente cuando es poco probable que lo consiga.

La biblioteca está escrita en C y está diseñada para cumplir con las normas ISO C90 y MISRA C:2012. La biblioteca no depende de ninguna biblioteca adicional que no sea la biblioteca C estándar y no tiene asignación de pilas, lo que la hace adecuada para microcontroladores de IoT y es totalmente portátil a otras plataformas.

Esta biblioteca se puede utilizar libremente y se distribuye bajo la licencia de código abierto de MIT.

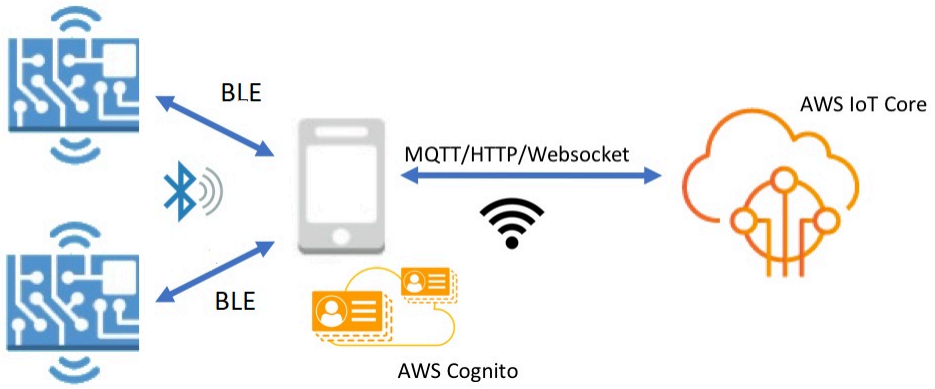
Tamaño de código de backoffAlgorithm (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
backoff_algorithm.c	0,1 K	0,1 K
Estimaciones totales	0,1 K	0,1 K

Biblioteca de Bluetooth de bajo consumo

> **Important**
>
> Esta biblioteca está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Información general

FreeRTOS admite la publicación y suscripción a temas de Message Queuing Telemetry Transport (MQTT) sobre Bluetooth de bajo consumo (BLE) a través de un dispositivo proxy, como un teléfono móvil. Con la biblioteca de Bluetooth de bajo consumo (BLE) de FreeRTOS, el microcontrolador puede comunicarse de forma segura con el agente de MQTT de AWS IoT.

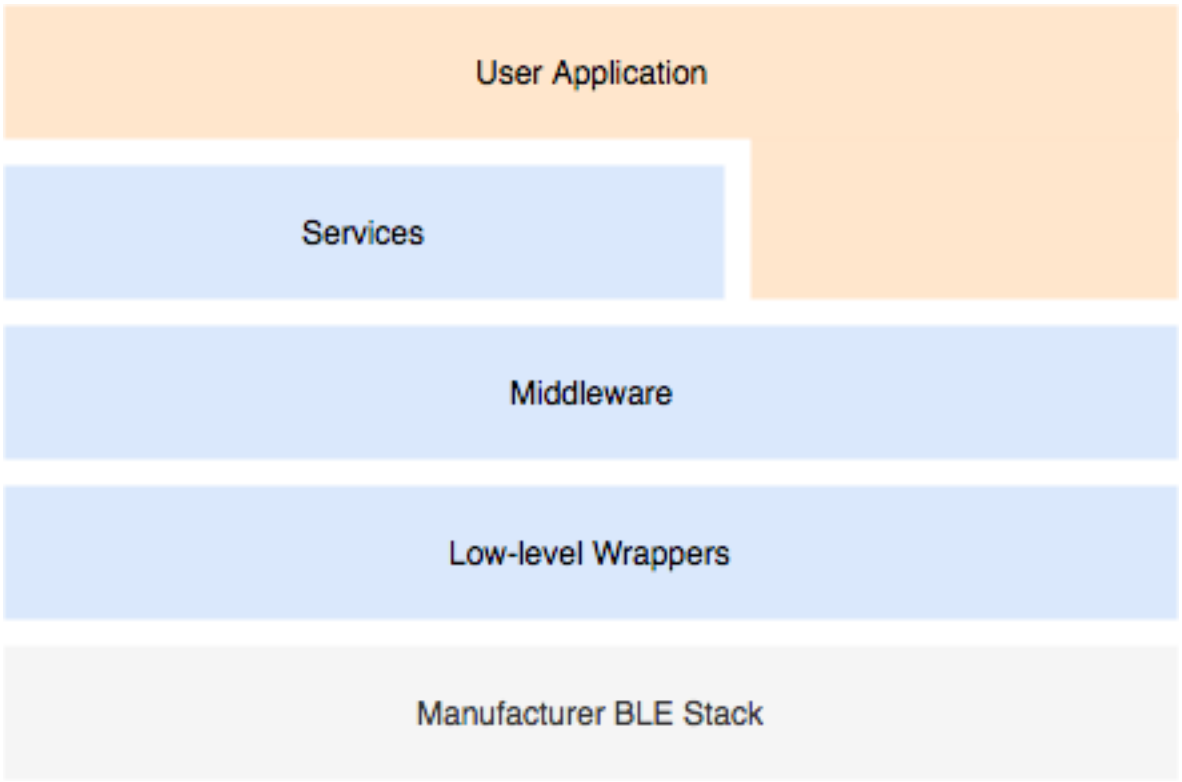


Con los SDK para móviles de dispositivos Bluetooth de FreeRTOS, puede crear aplicaciones móviles nativas que se comuniquen con las aplicaciones integradas en su microcontrolador a través de BLE. Para obtener más información acerca de los SDK para móviles, consulte SDK para móviles para dispositivos Bluetooth de FreeRTOS.

La biblioteca de BLE de FreeRTOS incluye servicios para configurar redes Wi-Fi, transferir grandes cantidades de datos y proporcionar abstracciones de red a través de BLE. La biblioteca de BLE de FreeRTOS también incluye middleware y algunas API de nivel inferior para un control más directo sobre su pila BLE.

Arquitectura

La biblioteca de BLE de FreeRTOS se compone de tres capas: servicios, middleware y contenedores de bajo nivel.



Servicios

La capa de servicios BLE de FreeRTOS consta de cuatro servicios de atributos genéricos (GATT) que aprovechan las API de middleware:

- Información de dispositivo
- Aprovisionamiento Wi-Fi
- Abstracción de red
- Transferencia de objetos grandes

Información de dispositivo

El servicio de Información de dispositivo recopila información sobre su microcontrolador, que incluye:

- La versión de FreeRTOS que su dispositivo está utilizando.
- El punto de enlace de AWS IoT de la cuenta para la que está registrado el dispositivo.
- Unidad de transmisión máxima (MTU) de Bluetooth de bajo consumo.

Aprovisionamiento Wi-Fi

El servicio de Aprovisionamiento Wi-Fi permite microcontroladores con capacidades Wi-Fi para hacer lo siguiente:

- Crear una lista de redes en el rango de alcance.
- Guardar redes y credenciales de red en la memoria flash.
- Establecer prioridad de red.
- Eliminar redes y credenciales de red de la memoria flash.

Abstracción de red

El servicio de abstracción de red abstrae el tipo de conexión de red para las aplicaciones. Una API común interactúa con la pila de hardware de Wi-Fi, Ethernet y Bluetooth de bajo consumo de su dispositivo, lo que permite que una aplicación sea compatible con varios tipos de conexión.

Transferencia de objetos grandes

El servicio de transferencia de objetos grandes envía y recibe datos de un cliente. Otros servicios, como Aprovisionamiento Wi-Fi y Abstracción de red, utilizan el servicio de Transferencia de objetos grandes para enviar y recibir datos. También puede utilizar la API de transferencia de objetos grandes para interactuar con el servicio directamente.

MQTT sobre BLE

MQTT sobre BLE contiene el perfil de GATT para crear un servicio proxy de MQTT sobre BLE. El servicio proxy de MQTT permite que un cliente de MQTT se comunique con el agente de MQTT de AWS a través de un dispositivo de puerta de enlace. Por ejemplo, puede usar el servicio proxy para conectar un dispositivo que ejecute FreeRTOS a MQTT de AWS a través de una aplicación para teléfonos inteligentes. El dispositivo BLE es el servidor del GATT y expone los servicios y las características del dispositivo de puerta de enlace. El servidor del GATT utiliza estos servicios y características expuestos para realizar operaciones de MQTT con la nube para ese dispositivo. Para obtener más detalles, consulte Apéndice A: perfil de GATT de MQTT sobre BLE .

Middleware

El middleware de Bluetooth de bajo consumo de FreeRTOS es una abstracción de las API de bajo nivel. Las API de middleware componen una interfaz más fácil de utilizar para la pila de Bluetooth de bajo consumo.

Con las API de middleware, puede registrar varias devoluciones de llamada, en varias capas, en un único evento. La inicialización del middleware de Bluetooth de bajo consumo también inicializa los servicios y comienza la publicidad.

Suscripción de devolución de llamada flexible

Supongamos que su hardware de Bluetooth de bajo consumo se desconecta y el servicio de MQTT sobre Bluetooth de bajo consumo necesita detectar la desconexión. Una aplicación que escribiera también podría necesitar detectar el mismo evento de desconexión. El middleware de Bluetooth de bajo consumo puede enrutar el evento a diferentes partes del código en el que haya registrado devoluciones de llamada, sin que la capas superiores compitan por recursos de nivel inferior.

Contenedores de bajo nivel

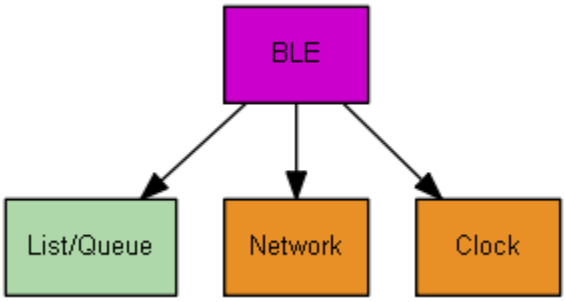
Los contenedores de Bluetooth de bajo consumo de FreeRTOS de bajo nivel son una abstracción de la pila de Bluetooth de bajo consumo del fabricante. Los contenedores de bajo nivel ofrecen un conjunto común de API para control directo sobre el hardware. Las API de bajo nivel optimizan el uso de RAM, pero su funcionalidad es limitada.

Utilice las API de servicio de Bluetooth de bajo consumo para interactuar con los servicios de Bluetooth de bajo consumo. Las API de servicio exigen más recursos que las API de bajo nivel.

Dependencias y requisitos

La biblioteca de Bluetooth de bajo consumo tiene las siguientes dependencias directas:

- Biblioteca de contenedores lineales
- Una capa de plataforma que interactúa con el sistema operativo para la administración de subprocesos, temporizadores, funciones de reloj y acceso a la red.



Solo el servicio de aprovisionamiento Wi-Fi tiene dependencias de la biblioteca de FreeRTOS:

Servicio de GATT	Dependencia
Aprovisionamiento Wi-Fi	Biblioteca wifi

Para comunicarse con el agente de MQTT de AWS IoT, debe disponer de una cuenta de AWS y debe registrar sus dispositivos como objetos de AWS IoT. Para obtener más información sobre la configuración, consulte la Guía para desarrolladores de AWS IoT.

Bluetooth de bajo consumo de FreeRTOS utiliza Amazon Cognito para la autenticación de usuarios en su dispositivo móvil. Para utilizar los servicios proxy de MQTT, debe crear una identidad y grupos de usuarios de Amazon Cognito. Cada identidad de Amazon Cognito debe tener asociada la política apropiada. Para obtener más información, consulte la Guía para desarrolladores de Amazon Cognito.

Archivo de configuración de la biblioteca

Las aplicaciones que usan el servicio de MQTT sobre Bluetooth de bajo consumo de FreeRTOS deben proporcionar un archivo de encabezado `iot_ble_config.h`, en el que se definen los parámetros de configuración. Los parámetros de configuración sin definir toman los valores predeterminados especificados en `iot_ble_config_defaults.h`.

Algunos parámetros de configuración importantes son:

`IOT_BLE_ADD_CUSTOM_SERVICES`

Permite a los usuarios crear sus propios servicios.

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

Permite a los usuarios personalizar la publicidad y examinar los mensajes de respuesta.

Para obtener más información, consulte la Referencia de la API de Bluetooth de bajo consumo.

Optimización

Al optimizar el rendimiento de la placa, tenga en cuenta lo siguiente:

- Las API de bajo nivel usan menos RAM, pero ofrecen una funcionalidad limitada.
- Puede establecer el parámetro `bleconfigMAX_NETWORK` en el archivo de encabezado `iot_ble_config.h` en un valor inferior para reducir la cantidad de pila consumida.

- Puede aumentar el tamaño de MTU a su valor máximo para limitar el almacenamiento en búfer de mensajes y hacer que el código se ejecute más rápido y consuma menos RAM.

Restricciones de uso

De forma predeterminada, la biblioteca de Bluetooth de bajo consumo de FreeRTOS establece la propiedad `eBTpropertySecureConnectionOnly` en TRUE, lo que coloca el dispositivo en un modo de Solo conexiones seguras. Como se especifica en Bluetooth Core Specification v5.0, Vol 3, parte C, 10.2.4, cuando un dispositivo se encuentra en modo de Solo conexiones seguras, se requiere el nivel de modo de seguridad LE 1 más alto, el nivel 4 para acceder a cualquier atributo que tenga permisos más altos que el nivel del modo de seguridad LE 1 más bajo, el nivel 1. En el nivel 4 del modo de seguridad LE 1, un dispositivo debe tener capacidades de entrada y salida para la comparación numérica.

Estos son los modos compatibles y sus propiedades asociadas:

Modo 1, nivel 1 (sin seguridad)

```
/* Disable numeric comparison */
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON       ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                    ( eBTIONone )
#define IOT_BLE_ENCRYPTION_REQUIRED              ( 0 )
```

Modo 1, nivel 2 (emparejamiento no autenticado con cifrado)

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON       ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                    ( eBTIONone )
```

Modo 1, nivel 3 (emparejamiento autenticado con cifrado)

Este modo no se admite.

Modo 1, nivel 4 (emparejamiento de conexiones seguras LE autenticadas con cifrado)

Este modo se admite de forma predeterminada.

Para obtener más información acerca de los modos de seguridad LE, consulte Bluetooth Core Specification v5.0, Vol 3, parte C, 10.2.1.

Inicialización

Si la aplicación interactúa con la pila de Bluetooth de bajo consumo a través de middleware, solo tiene que inicializar el middleware. El middleware se encarga de inicializar las capas inferiores de la pila.

Middleware

Para inicializar el middleware

1. Inicialice cualquier controlador de hardware de Bluetooth de bajo consumo antes de llamar a la API de middleware de Bluetooth de bajo consumo.
2. Habilite Bluetooth de bajo consumo.
3. Inicialice el middleware con `IotBLE_Init()`.

> **Note**
>
> Este paso de inicialización no es necesario si está ejecutando demostraciones de AWS. El administrador de red se encarga de la inicialización de la demostración, que se encuentra en *freertos*`/demos/network_manager`.

API de bajo nivel

Si no desea utilizar los servicios de GATT de Bluetooth de bajo consumo de FreeRTOS, puede omitir el middleware e interactuar directamente con las API de bajo nivel para ahorrar recursos.

Para inicializar las API de bajo nivel

1. Inicialice cualquier controlador de hardware de Bluetooth de bajo consumo antes de llamar a las API. La inicialización del controlador no forma parte de las API de bajo nivel de Bluetooth de bajo consumo.
2. La API de bajo nivel de Bluetooth de bajo consumo dispone de una llamada de habilitación/deshabilitación a la pila de Bluetooth de bajo consumo para optimizar la alimentación y los recursos. Antes de llamar a las API, debe habilitar Bluetooth de bajo consumo.

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
```

```
xStatus = pxIface->pxEnable( 0 );
```

3. El administrador de Bluetooth contiene API que son comunes para Bluetooth de bajo consumo y Bluetooth Classic. Las devoluciones de llamada para el administrador común se deben inicializar en segundo lugar.

```
xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
```

4. El adaptador de Bluetooth de bajo consumo se adapta sobre la API común. Debe inicializar sus devoluciones de llamada de la misma forma que se inicializa la API común.

```
xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * )
 xBTInterface.pxBTInterface->pxGetLeAdapter();
xStatus = xBTInterface.pxBTLeAdapterInterface-
>pxBleAdapterInit( &xBTBleAdapterCb );
```

5. Registre su nueva aplicación de usuario.

```
xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
```

6. Inicialice las devoluciones de llamadas a los servidores de GATT.

```
xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
 xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

   Después de inicializar el adaptador de Bluetooth de bajo consumo, puede agregar un servidor de GATT. Solo puede registrar los servidor de GATT de uno en uno.

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. Establezca las propiedades de la aplicación solo como conexión segura y tamaño de MTU.

```
xStatus = xBTInterface.pxBTInterface-
>pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

Referencia de la API

Para ver una referencia completa de la API, consulte la Referencia de la API de Bluetooth de bajo consumo.

Ejemplo de uso

Los siguientes ejemplos muestran cómo utilizar la biblioteca de Bluetooth de bajo consumo para la publicidad y la creación de nuevos servicios. Para aplicaciones de demostración de Bluetooth de bajo consumo de FreeRTOS, consulte Aplicaciones de demostración de Bluetooth de bajo consumo.

Publicidad

1. En la aplicación, defina el UUID de la publicidad:

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBTuuidType128
};
```

2. A continuación, defina la función de devolución de llamada `IotBle_SetCustomAdvCb`:

```
void IotBle_SetCustomAdvCb( IotBleAdvertisementParams_t * pAdvParams,
 IotBleAdvertisementParams_t * pScanParams)
{
    memset(pAdvParams, 0, sizeof(IotBleAdvertisementParams_t));
    memset(pScanParams, 0, sizeof(IotBleAdvertisementParams_t));

    /* Set advertisement message */
    pAdvParams->pUUID1 = &_advUUID;
    pAdvParams->nameType = BTGattAdvNameNone;

    /* This is the scan response, set it back to true. */
    pScanParams->setScanRsp = true;
    pScanParams->nameType = BTGattAdvNameComplete;
}
```

   Esta devolución de llamada envía el UUID en el mensaje de publicidad y el nombre completo en la respuesta al examen.

3. Abra `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` y establezca `IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG` en 1. Esto activa la devolución de llamada `IotBle_SetCustomAdvCb`.

Adición de un nuevo servicio

Para ver ejemplos completos de servicios, consulte *freertos*`/.../ble/services`.

1. Cree los UUID para los descriptores y características del servicio:

```
#define xServiceUUID_TYPE \
{\
    .uu.uu128 = gattDemoSVC_UUID, \
    .ucType   = eBTuuidType128 \
}
#define xCharCounterUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_COUNTER_UUID,\
    .ucType   = eBTuuidType128\
}
#define xCharControlUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_CONTROL_UUID,\
    .ucType   = eBTuuidType128\
}
#define xClientCharCfgUUID_TYPE \
{\
    .uu.uu16 = gattDemoCLIENT_CHAR_CFG_UUID,\
    .ucType  = eBTuuidType16\
}
```

2. Cree un búfer para registrar los controladores de características y descriptores.

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. Cree la tabla de atributos. Para ahorrar algo de RAM, defina la tabla como una `const`.

> **Important**
>
> Siempre cree los atributos en orden, con el servicio como el primer atributo.

```
static const BTAttribute_t pxAttributeTable[] = {
     {
          .xServiceUUID =  xServiceUUID_TYPE
     },
    {
          .xAttributeType = eBTDbCharacteristic,
          .xCharacteristic =
          {
               .xUuid = xCharCounterUUID_TYPE,
               .xPermissions = ( IOT_BLE_CHAR_READ_PERM ),
               .xProperties = ( eBTPropRead | eBTPropNotify )
           }
     },
     {
          .xAttributeType = eBTDbDescriptor,
          .xCharacteristicDescr =
          {
              .xUuid = xClientCharCfgUUID_TYPE,
              .xPermissions = ( IOT_BLE_CHAR_READ_PERM | IOT_BLE_CHAR_WRITE_PERM )
           }
     },
    {
          .xAttributeType = eBTDbCharacteristic,
          .xCharacteristic =
          {
               .xUuid = xCharControlUUID_TYPE,
               .xPermissions = ( IOT_BLE_CHAR_READ_PERM | IOT_BLE_CHAR_WRITE_PERM
  ),
               .xProperties = ( eBTPropRead | eBTPropWrite )
           }
     }
};
```

4. Cree una matriz de devoluciones de llamada. Esta matriz de devoluciones de llamada debe seguir el mismo orden que la matriz de la tabla descrita anteriormente.

   Por ejemplo, si `vReadCounter` se acciona cuando se accede a `xCharCounterUUID_TYPE` y `vWriteCommand` se acciona cuando se accede a `xCharControlUUID_TYPE`, defina la matriz tal y como se indica a continuación:

```
static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes]
 =
    {
  NULL,
  vReadCounter,
  vEnableNotification,
  vWriteCommand
};
```

5. Cree el servicio:

```
static const BTService_t xGattDemoService =
{
  .xNumberOfAttributes = egattDemoNbAttributes,
  .ucInstId = 0,
  .xType = eBTServiceTypePrimary,
  .pusHandlesBuffer = usHandlesBuffer,
  .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. Llame a la API `IotBle_CreateService` con la estructura que ha creado en el paso anterior. El middleware sincroniza la creación de todos los servicios, por lo que cualquier servicio nuevo ya deberá haberse definido cuando se activa la devolución de llamada `IotBle_AddCustomServicesCb`.

   a. En `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h`, establezca `IOT_BLE_ADD_CUSTOM_SERVICES` en 1.

   b. Cree IotBle_AddCustomServicesCb en su aplicación:

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

Portabilidad

Periférico de entrada y salida de usuario

Una conexión segura requiere entrada y salida para comparación numérica. El evento de `eBLENumericComparisonCallback` pueden registrarse con el administrador de eventos:

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

El periférico deben mostrar la clave de acceso numérica y tomar el resultado de la comparación como una entrada.

Portabilidad de implementaciones de API

Para realizar la portabilidad de FreeRTOS a un nuevo destino, debe implementar algunas API para el servicio de aprovisionamiento Wi-Fi y la funcionalidad de Bluetooth de bajo consumo.

API de Bluetooth de bajo consumo

Para utilizar el middleware de Bluetooth de bajo consumo de FreeRTOS, debe implementar algunas API.

API común entre GAP para Bluetooth Classic y GAP para Bluetooth de bajo consumo

- `pxBtManagerInit`
- `pxEnable`
- `pxDisable`
- `pxGetDeviceProperty`
- `pxSetDeviceProperty` (Todas las opciones son obligatorios esperan `eBTpropertyRemoteRssi` y `eBTpropertyRemoteVersionInfo`)
- `pxPair`
- `pxRemoveBond`
- `pxGetConnectionState`
- `pxPinReply`
- `pxSspReply`

- pxGetTxpower
- pxGetLeAdapter
- pxDeviceStateChangedCb
- pxAdapterPropertiesCb
- pxSspRequestCb
- pxPairingStateChangedCb
- pxTxPowerCb

API específicas para GAP para Bluetooth de bajo consumo

- pxRegisterBleApp
- pxUnregisterBleApp
- pxBleAdapterInit
- pxStartAdv
- pxStopAdv
- pxSetAdvData
- pxConnParameterUpdateRequest
- pxRegisterBleAdapterCb
- pxAdvStartCb
- pxSetAdvDataCb
- pxConnParameterUpdateRequestCb
- pxCongestionCb

Servidor de GATT

- pxRegisterServer
- pxUnregisterServer
- pxGattServerInit
- pxAddService
- pxAddIncludedService

- `pxAddCharacteristic`
- `pxSetVal`
- `pxAddDescriptor`
- `pxStartService`
- `pxStopService`
- `pxDeleteService`
- `pxSendIndication`
- `pxSendResponse`
- `pxMtuChangedCb`
- `pxCongestionCb`
- `pxIndicationSentCb`
- `pxRequestExecWriteCb`
- `pxRequestWriteCb`
- `pxRequestReadCb`
- `pxServiceDeletedCb`
- `pxServiceStoppedCb`
- `pxServiceStartedCb`
- `pxDescriptorAddedCb`
- `pxSetValCallbackCb`
- `pxCharacteristicAddedCb`
- `pxIncludedServiceAddedCb`
- `pxServiceAddedCb`
- `pxConnectionCb`
- `pxUnregisterServerCb`
- `pxRegisterServerCb`

Para obtener más información acerca de la portabilidad de la biblioteca de Bluetooth de bajo consumo de FreeRTOS para su plataforma, consulte la sección sobre Portabilidad de Bluetooth de bajo consumo en la Guía de portabilidad de FreeRTOS.

SDK para móviles para dispositivos Bluetooth de FreeRTOS

> **Important**
>
> Esta biblioteca está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Puede utilizar los SDK para móviles para dispositivos Bluetooth de FreeRTOS para crear aplicaciones móviles que interactúen con su microcontrolador a través de Bluetooth de bajo consumo. Los SDK para móviles también pueden comunicarse con los servicios de AWS utilizando Amazon Cognito para la autenticación de usuarios.

SDK para Android para dispositivos Bluetooth de FreeRTOS

Utilice el SDK para Android para dispositivos Bluetooth de FreeRTOS para crear aplicaciones móviles para Android que interactúen con su microcontrolador a través de Bluetooth de bajo consumo. El SDK está disponible en GitHub.

Para instalar el SDK para Android para dispositivos Bluetooth FreeRTOS, siga las instrucciones de “Configuración del SDK” incluidas el archivo README.md del proyecto.

Para obtener información acerca de cómo configurar y ejecutar la aplicación móvil de demostración que se incluye con el SDK, consulte Requisitos previos y Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS.

SDK para iOS para dispositivos Bluetooth de FreeRTOS

Utilice el SDK para iOS para dispositivos Bluetooth de FreeRTOS para crear aplicaciones móviles para iOS que interactúen con su microcontrolador a través de Bluetooth de bajo consumo. El SDK está disponible en GitHub.

Para instalar el SDK de iOS

1. Instale CocoaPods:

```
$ gem install cocoapods
$ pod setup
```

> **Note**
>
> Es posible que necesite usar `sudo` para instalar CocoaPods.

2. Instale el SDK con CocoaPods (agregue esto a su podfile):

```
$ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
```

Para obtener información acerca de cómo configurar y ejecutar la aplicación móvil de demostración que se incluye con el SDK, consulte Requisitos previos y Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS.

Apéndice A: perfil de GATT de MQTT sobre BLE

Detalles del servicio GATT

MQTT sobre BLE utiliza una instancia del servicio GATT de transferencia de datos para enviar mensajes de representación concisa de objetos binarios (CBOR) de MQTT entre el dispositivo FreeRTOS y el dispositivo proxy. El servicio de transferencia de datos presenta ciertas características que ayudan a enviar y recibir datos sin procesar a través del protocolo GATT de BLE. También gestiona la fragmentación y el ensamblaje de cargas útiles superiores al tamaño de la unidad máxima de transferencia (MTU) del BLE.

UUID del servicio

`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00`

Instancias de servicio

Se crea una instancia del servicio GATT para cada sesión de MQTT con el agente. Cada servicio tiene un UUID único (dos bytes) que identifica su tipo. Cada instancia individual se diferencia por el ID de la instancia.

Cada servicio se instancia como un servicio principal en cada dispositivo del servidor BLE. Puede crear varias instancias del servicio en un dispositivo determinado. El tipo de servicio proxy de MQTT tiene un UUID único.

Características

Formato de contenido de característica: CBOR

Tamaño máximo del valor de la característica: 512 bytes

Característica	Requisito	Propiedades obligatorias	Propiedades opcionales	Permisos de seguridad	Descripción breve	UUID (Identificador único universal)
Controlar	M	Escritura	Ninguno	Escritura necesita cifrado	Se utiliza para iniciar y detener el proxy de MQTT.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF01`
TXMessage	M	Lectura, notificación	Ninguno	Lectura necesita cifrado	Se utiliza para enviar una notificación que contiene un mensaje a un agente a través de un proxy.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF02`
RXMessage	M	Lectura, escritura sin respuesta	Ninguno	Lectura, escritura necesita cifrado	Se utiliza para recibir un mensaje de un agente a través de un proxy.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF03`

Característica	Requisito	Propiedades obligatorias	Propiedades opcionales	Permisos de seguridad	Descripción breve	UUID (Identificador único universal)
TXLargeMessage	M	Lectura, notificación	Ninguno	Lectura necesita cifrado	Se utiliza para enviar un mensaje grande (mensaje > tamaño de MTU de BLE) a un agente a través de un proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF04
RXLargeMessage	M	Lectura, escritura sin respuesta	Ninguno	Lectura, escritura necesita cifrado	Se utiliza para enviar un mensaje grande (mensaje > tamaño de MTU de BLE) a un agente a través de un proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF05

Requisitos del procedimiento de GATT

Valores de característica de lectura	Obligatorio
Valores largos de característica de lectura	Obligatorio
Valores de característica de escritura	Obligatorio
Valores largos de característica de escritura	Obligatorio
Descriptores de característica de lectura	Obligatorio
Descriptores de característica de escritura	Obligatorio
Notificaciones	Obligatorio
Indicaciones	Obligatorio

Tipos de mensajes

Se intercambian los siguientes tipos de mensajes.

Tipo de mensaje	Mensaje	Asignación con estos pares de claves/valores
0x01	CONNECT	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (1) • Clave = “d”, valor = Tipo 3, Cadena de texto, Identificador de cliente para la sesión • Clave = “a”, valor = Tipo 3, Cadena de texto, Punto de conexión de agente para la sesión

Tipo de mensaje	Mensaje	Asignación con estos pares de claves/valores
		• Clave = “c”, valor = Valor simple de tipo verdadero/falso
0x02	CONNACK	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (2) • Clave = “s”, valor = Entero de tipo 0, Código de estado
0x03	PUBLISH	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (3) • Clave = “u”, valor = Tipo 3, Cadena de texto, Tema para publicación • Clave = “n”, valor = Tipo 0, Entero, QoS para publicación • Clave = “i”, valor = Tipo 0, Entero, Identificador de mensaje, Solo para publicaciones de QoS 1 • Clave = “k”, valor = Tipo 2, Cadena de bytes, Carga útil para publicación

Tipo de mensaje	Mensaje	Asignación con estos pares de claves/valores
0x04	PUBACK	• Se envía solo para mensajes de QoS 1. • Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (4) • Clave = “i”, valor = Tipo 0, Entero, Identificador de mensaje
0x08	SUBSCRIBE	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (8) • Clave = “v”, valor = Tipo 4, Matriz de cadenas de texto, temas para suscripción • Clave = “o”, valor = Tipo 4, Matriz de enteros, QoS para suscripción • Clave = “i”, valor = Tipo 0, Entero, Identificador de mensaje
0x09	SUBACK	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (9) • Clave = “i”, valor = Tipo 0, Entero, Identificador de mensaje • Clave = “s”, valor = Tipo 0, Entero, Código de estado para suscripción

Tipo de mensaje	Mensaje	Asignación con estos pares de claves/valores
0X0A	UNSUBSCRIBE	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (10) • Clave = “v”, valor = Tipo 4, Matriz de cadenas de texto, temas para cancelación de suscripción • Clave = “i”, valor = Tipo 0, Entero, Identificador de mensaje
0x0B	UNSUBACK	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (11) • Clave = “i”, valor = Tipo 0, Entero, Identificador de mensaje • Clave = “s”, valor = Tipo 0, Entero, Código de estado para cancelación de suscripción
0X0C	PINGREQ	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (12)
0x0D	PINGRESP	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (13)
0x0E	DISCONNNECT	• Clave = “w”, valor = Entero de tipo 0, Tipo de mensaje (14)

Características de transferencia de carga útil grande

TXLargeMessage

El dispositivo utiliza TXLargeMessage para enviar una carga útil grande que es mayor que el tamaño de MTU negociado para la conexión BLE.

- El dispositivo envía los primeros bytes de MTU de la carga útil como una notificación a través de la característica.
- El proxy envía una solicitud de lectura sobre esta característica para los bytes restantes.
- El dispositivo envía hasta el tamaño de la MTU o los bytes restantes de la carga útil, lo que sea menor. Cada vez, aumenta el desplazamiento leído en función del tamaño de la carga útil enviada.
- El proxy seguirá leyendo la característica hasta que obtenga una carga útil de longitud cero o una carga útil inferior al tamaño de la MTU.
- Si el dispositivo no recibe una solicitud de lectura dentro de un tiempo de espera especificado, se produce un error en la transferencia y el proxy y la puerta de enlace liberan el búfer.
- Si el proxy no recibe una solicitud de lectura dentro de un tiempo de espera especificado, se produce un error en la transferencia y el proxy libera el búfer.

RXLargeMessage

El dispositivo utiliza RXLargeMessage para recibir una carga útil grande que es mayor que el tamaño de MTU negociado para la conexión BLE.

- El proxy escribe los mensajes, hasta el tamaño de la MTU, uno por uno, utilizando la función de escritura con respuesta según esta característica.
- El dispositivo almacena el mensaje en búfer hasta que recibe una solicitud de escritura con una longitud cero o una longitud inferior al tamaño de la MTU.
- Si el dispositivo no recibe una solicitud de escritura dentro de un tiempo de espera especificado, se produce un error en la transferencia y el dispositivo libera el búfer.
- Si el proxy no recibe una solicitud de escritura dentro de un tiempo de espera especificado, se produce un error en la transferencia y el proxy libera el búfer.

Biblioteca de interfaces móviles

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

La biblioteca Cellular Interface implementa una API simple y unificada que oculta la complejidad de los comandos AT específicos del módem celular y expone una interfaz similar a un socket a los programadores de C.

La mayoría de los módems móviles implementan más o menos los comandos AT definidos por el estándar TS v27.007 de 3GPP. Este proyecto proporciona una implementación de dichos comandos AT estándar en un componente común reutilizable. Las tres bibliotecas de interfaces móviles de este proyecto aprovechan ese código común. La biblioteca de cada módem solo implementa los comandos AT específicos del proveedor y, a continuación, expone la API completa de la biblioteca de interfaces móviles.

El componente común que implementa el estándar TS v27.007 de 3GPP se ha diseñado de acuerdo con los siguientes criterios de calidad del código:

- Las puntuaciones de complejidad de GNU no superan 8
- Estándar de codificación MISRA C:2012. Cualquier desviación del estándar se documenta en los comentarios del código fuente marcados con “coverity”.

Dependencias y requisitos

No existe una dependencia directa para la biblioteca de interfaces celulares. Sin embargo, Ethernet, Wi-Fi y móvil no pueden coexistir en la pila de la red de FreeRTOS. Los desarrolladores deben elegir una de las interfaces de red para integrarla con la biblioteca de socket seguros.

Portabilidad

Para obtener más información acerca de la portabilidad de la biblioteca interfaces móviles a su plataforma, consulte Portabilidad de la biblioteca de interfaces móviles en la Guía de portabilidad de FreeRTOS.

Uso de memoria

Tamaño de código de la biblioteca de interfaces móviles (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
cellular_3gpp_api.c	6,3 K	5,7 K
cellular_3gpp_urc_handler.c	0,9 K	0,8 K
cellular_at_core.c	1,4 K	1,2 K
cellular_common_api.c	0,5 K	0,5 K
cellular_common.c	1,6 K	1,4 K
cellular_pkthandler.c	1,4 K	1,2 K
cellular_pktio.c	1,8 K	1,6 K
Estimaciones totales	13,9 K	12,4K

Introducción

Descarga del código fuente

El código fuente se puede descargar como parte de las bibliotecas de FreeRTOS o solo.

Para clonar la biblioteca de Github mediante HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Uso de SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Estructura de carpeta

En la raíz de este repositorio, verá estas carpetas:

- `source` : código común reutilizable que implementa los comandos AT estándar definidos por TS v27.007 de 3GPP
- `doc` : documentación
- `test` : prueba unitaria y cbmc
- `tools` : herramientas para el análisis estático de Coverity y CMock

Configuración y creación de la biblioteca

La biblioteca de interfaces móviles debe crearse como parte de una aplicación. Para ello, debe proporcionar ciertas configuraciones. El proyecto Freertos_Cellular_Interface_Windows_Simulator proporciona un ejemplo de cómo configurar la creación. Encontrará más información en las Referencias de la API para móviles.

Consulte la página Interfaz móvil para obtener más información.

Integración de la biblioteca de interfaces móviles con plataformas MCU

La biblioteca de interfaces móviles se ejecuta en las MCU mediante una interfaz abstracta, la interfaz de comunicación, para comunicarse con los módems móviles. También se debe implementar una interfaz de comunicación en la plataforma MCU. Las implementaciones más comunes de la interfaz de comunicación se comunican a través del hardware UART, pero también se pueden implementar a través de otras interfaces físicas, como SPI. La documentación sobre la interfaz de comunicación se encuentra en las referencias de la API de la biblioteca de interfaces móviles. Están disponibles los siguientes ejemplos de implementaciones de la interfaz de comunicación:

- Interfaz de comunicación del simulador FreeRTOS para Windows
- Interfaz de comunicación UART de E/S común de FreeRTOS
- Interfaz de comunicación de la placa de detección STM32 L475
- Interfaz de comunicación de la placa del hub del sensor Sierra

E/S común

> **Important**
>
> Esta biblioteca está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto

FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Información general

En general, los controladores de dispositivo son independientes del sistema operativo subyacente y son específicos de una configuración de hardware determinada. Una capa de abstracción de hardware (HAL) proporciona una interfaz común entre controladores y código de aplicación de nivel superior. La capa HAL abstrae los detalles de cómo funciona un controlador específico y proporciona una API uniforme para controlar dichos dispositivos. Puede utilizar las mismas API para obtener acceso a varios controladores de dispositivo a través de tarjetas de referencia basadas en varios microcontroladores (MCU).

La E/S común de FreeRTOS actúa como esta capa de abstracción de hardware. Proporciona un conjunto de API estándar para obtener acceso a dispositivos serie comunes en placas de referencia compatibles. Estas API comunes se comunican e interactúan con estos periféricos y permiten que el código funcione en todas las plataformas. Sin E/S común, el código para dispositivos de bajo nivel es específico del proveedor de hardware.

Periféricos admitidos

- UART
- SPI
- I2C

Características admitidas

- Lectura y escritura sincrónicas - La función no regresa hasta que se transfiere la cantidad de datos solicitada.
- Lectura y escritura asíncronas - La función regresa inmediatamente y la transferencia de datos se realiza de forma asíncrona. Cuando se completa la acción, se invoca una devolución de llamada de usuario registrado.

Código específico del periférico

- I2C - Combina varias operaciones en una sola transacción. Se utiliza para acciones de escritura y luego lectura en una sola transacción.

- SPI - Transfiere datos entre la principal y secundaria, lo que significa que la escritura y la lectura se realizan simultáneamente.

Portabilidad

Para obtener más información, consulte la Guía de portabilidad de FreeRTOS.

Biblioteca de AWS IoT Device Defender

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

Puede usar la AWS IoT Device Defender biblioteca para enviar métricas de seguridad desde sus dispositivos de IoT aAWS IoT Device Defender. Puede utilizar AWS IoT Device Defender para monitorear continuamente estas métricas de seguridad de los dispositivos para detectar desviaciones con respecto a lo que haya definido como comportamiento apropiado para cada dispositivo. Si algo no está bien, AWS IoT Device Defender envía una alerta para que pueda tomar medidas para solucionar el problema. Las interacciones con AWS IoT Device Defender utilizan MQTT, un protocolo ligero de publicación y suscripción. Esta biblioteca proporciona una API para componer y reconocer las cadenas de temas MQTT que utiliza AWS IoT Device Defender.

Para obtener más información, consulte AWS IoT Device Defender en la Guía para desarrolladores de AWS IoT.

La biblioteca está escrita en C y está diseñada para cumplir con las normas ISO C90 y MISRA C:2012. La biblioteca no depende de ninguna biblioteca adicional que no sea la biblioteca C estándar. Tampoco tiene dependencias de plataforma, como el subprocesamiento o la sincronización. Se puede usar con cualquier biblioteca MQTT y cualquier biblioteca JSON o CBOR. La biblioteca tiene pruebas que muestran un uso seguro de la memoria y la ausencia de asignación de pilas, lo que la hace adecuada para microcontroladores de IoT, pero también es totalmente portátil a otras plataformas.

La biblioteca AWS IoT Device Defender se puede utilizar libremente y se distribuye bajo la licencia de código abierto de MIT.

Tamaño de código de AWS IoT Device Defender (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
defender.c	1,1 K	0,6 K
Estimaciones totales	1,1 K	0,6 K

Biblioteca de detección de AWS IoT Greengrass

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Información general

Los dispositivos de microcontrolador utilizan la biblioteca de detección de AWS IoT Greengrass para detectar un núcleo de Greengrass en su red. Con las API de detección de AWS IoT Greengrass su dispositivo puede enviar mensajes a un núcleo de Greengrass después de encontrar el punto de enlace del núcleo.

Dependencias y requisitos

Para utilizar la biblioteca de detección de Greengrass, debe crear un objeto en AWS IoT, incluido un certificado y una política. Para obtener más información, consulte Introducción a AWS IoT.

Debe definir valores para las siguientes constantes en el archivo *freertos*`/demos/include/aws_clientcredential.h`:

`clientcredentialMQTT_BROKER_ENDPOINT`

El punto de enlace de AWS IoT.

`clientcredentialIOT_THING_NAME`

Es el nombre de su objeto de IoT.

`clientcredentialWIFI_SSID`

El SSID de su red wifi.

`clientcredentialWIFI_PASSWORD`

La contraseña de wifi.

`clientcredentialWIFI_SECURITY`

El tipo de seguridad utilizado por su red wifi.

También debe definir valores para las siguientes constantes en el archivo *`freertos`*`/demos/include/aws_clientcredential_keys.h`:

`keyCLIENT_CERTIFICATE_PEM`

El certificado PEM asociado a su objeto.

`keyCLIENT_PRIVATE_KEY_PEM`

El PEM de clave privada asociado a su objeto.

Debe tener un grupo de Greengrass y un dispositivo de núcleo configurado en la consola. Para obtener más información, consulte Introducción a AWS IoT Greengrass.

Aunque no se necesita una biblioteca de coreMQTT para la conectividad Greengrass, le recomendamos encarecidamente que la instale. La biblioteca se pueden utilizar para comunicarse con el núcleo de Greengrass después de que se haya detectado.

Referencia de la API

Para obtener una referencia de la API completa, consulte la Referencia de la API Greengrass.

Ejemplo de uso

Flujo de Greengrass

El dispositivo MCU inicia el proceso de detección solicitando un archivo JSON de AWS IoT que contiene los parámetros de conectividad del núcleo de Greengrass. Existen dos métodos para recuperar los parámetros de conectividad del núcleo de Greengrass del archivo JSON:

- La selección automática realiza iteraciones por todos los núcleos de Greengrass enumerados en el archivo JSON y se conecta al primero disponible.
- La selección manual utiliza la información en `aws_ggd_config.h` para conectarse al núcleo de Greengrass especificado.

Uso de la API de Greengrass

Todas las opciones de configuración predeterminadas para la API de Greengrass se definen en `aws_ggd_config_defaults.h`.

Si solo está presente un núcleo de Greengrass, llame a `GGD_GetGGCIPandCertificate` para solicitar el archivo JSON con información de conectividad del núcleo de Greengrass. Cuando se devuelve `GGD_GetGGCIPandCertificate`, el parámetro `pcBuffer` contiene el texto del archivo JSON. El parámetro `pxHostAddressData` contiene la dirección IP y el puerto del núcleo de Greengrass al que puede conectarse.

Para obtener más opciones de personalización, como la posibilidad de asignar certificados de forma dinámica, debe llamar a las siguientes API:

`GGD_JSONRequestStart`

Realiza una solicitud HTTP GET a AWS IoT para iniciar la solicitud de detección para detectar un núcleo de Greengrass. `GD_SecureConnect_Send` se utiliza para enviar la solicitud a AWS IoT.

`GGD_JSONRequestGetSize`

Obtiene el tamaño del archivo JSON de la respuesta HTTP.

`GGD_JSONRequestGetFile`

Obtiene la cadena de objeto JSON. `GGD_JSONRequestGetSize` y `GGD_JSONRequestGetFile` utilizan `GGD_SecureConnect_Read` para obtener los datos JSON del socket. Se debe llamar a `GGD_JSONRequestStart`, `GGD_SecureConnect_Send`, `GGD_JSONRequestGetSize` para recibir los datos JSON de AWS IoT.

`GGD_GetIPandCertificateFromJSON`

Extrae la dirección IP y el certificado del núcleo de Greengrass de los datos JSON. Puede activar la selección automática estableciendo `xAutoSelectFlag` en `True`. La selección automática encuentra el primer dispositivo de núcleo al que puede conectarse su dispositivo FreeRTOS. Para

conectarse a un núcleo de Greengrass, llame a la función `GGD_SecureConnect_Connect` y pase la dirección IP, el puerto y el certificado del dispositivo de núcleo. Para utilizar la selección manual, establezca los siguientes campos del parámetro `HostParameters_t`:

`pcGroupName`

El ID de grupo de Greengrass al que pertenece el núcleo. Puede utilizar el comando de la CLI `aws greengrass list-groups` para encontrar el ID de sus grupos de Greengrass.

`pcCoreAddress`

El ARN del núcleo de Greengrass al que se va a conectar.

Biblioteca coreHTTP

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Biblioteca de clientes HTTP C para dispositivos IoT pequeños (MCU o MPU pequeña)

Introducción

La biblioteca coreHTTP es una implementación de cliente de un subconjunto del estándar HTTP/1.1. El estándar HTTP proporciona un protocolo sin estado que se ejecuta sobre TCP/IP y se utiliza a menudo en sistemas de información de hipertexto distribuidos y colaborativos.

La biblioteca coreHTTP implementa un subconjunto del estándar de protocolo HTTP/1.1. Esta biblioteca se ha optimizado para reducir el consumo de memoria. La biblioteca proporciona una API totalmente sincrónica para que las aplicaciones puedan gestionar completamente su simultaneidad. Utiliza únicamente búferes fijos, de modo que las aplicaciones tienen el control total de su estrategia de asignación de memoria.

La biblioteca está escrita en C y está diseñada para cumplir con las normas ISO C90 y MISRA C:2012. Las únicas dependencias de la biblioteca son la biblioteca C estándar y la versión LTS (v12.19.1) de http-parser de Node.js. La biblioteca tiene pruebas que muestran un uso seguro de la memoria y la ausencia de asignación de pilas, lo que la hace adecuada para microcontroladores de IoT, pero también es totalmente portátil a otras plataformas.

Cuando utilice conexiones HTTP en aplicaciones de IoT, le recomendamos que utilice una interfaz de transporte segura, como una que utilice el protocolo TLS, tal y como se demuestra en Demostración de la autenticación mutua de coreHTTP.

Esta biblioteca se puede utilizar libremente y se distribuye bajo la licencia de código abierto de MIT.

Tamaño de código de coreHTTP (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
core_http_client.c	3,2 K	2,6 K
api.c (llhttp)	2,6 K	2,0 K
http.c (llhttp)	0,3 K	0,3 K
llhttp.c (llhttp)	17,9	15,9
Estimaciones totales	23,9 K	20,7 K

Biblioteca coreJSON

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

JSON (JavaScript Object Notation) es un formato de serialización de datos en lenguaje natural. Se usa ampliamente para intercambiar datos, como con el servicio Sombra de dispositivo de AWS IoT y forma parte de muchas API, como la API REST de GitHub. Ecma International mantiene JSON como estándar.

La biblioteca coreJSON proporciona un analizador que admite la búsqueda de claves aplicando la sintaxis de intercambio de datos JSON estándar ECMA-404. La biblioteca está escrita en C y está diseñada para cumplir con las normas ISO C90 y MISRA C:2012. Tiene pruebas que muestran

un uso seguro de la memoria y la ausencia de asignación de pilas, lo que la hace adecuada para microcontroladores de IoT, pero también es totalmente portátil a otras plataformas.

Uso de memoria

La biblioteca coreJSON usa una pila interna para rastrear las estructuras anidadas en un documento JSON. La pila existe mientras dure una sola llamada a una función; no se conserva. El tamaño de la pila se puede especificar definiendo la macro `JSON_MAX_DEPTH`, que por defecto es de 32 niveles. Cada nivel consume un solo byte.

Tamaño de código de coreJSON (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
core_json.c	2,9 K	2,4 K
Estimaciones totales	2,9 K	2,4 K

Biblioteca coreMQTT

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

La biblioteca coreMQTT es una implementación de cliente del estándar MQTT (Message Queue Telemetry Transport). El estándar MQTT proporciona un protocolo de mensajería ligero de publicación/suscripción (o PubSub) que se ejecuta sobre TCP/IP y se usa a menudo en casos de uso de máquina a máquina (M2M) e Internet de las cosas (IoT).

La biblioteca coreMQTT cumple con el estándar de protocolo MQTT 3.1.1. Esta biblioteca se ha optimizado para reducir el consumo de memoria. El diseño de esta biblioteca abarca diferentes casos de uso, desde plataformas con recursos limitados que utilizan únicamente mensajes QoS 0 MQTT PUBLISH hasta plataformas con muchos recursos que utilizan QoS 2 MQTT PUBLISH a través de conexiones TLS (Transport Layer Security). La biblioteca proporciona un menú de funciones

componibles, que se pueden elegir y combinar para adaptarse con precisión a las necesidades de un caso de uso concreto.

La biblioteca está escrita en C y está diseñada para cumplir con las normas ISO C90 y MISRA C:2012. Esta biblioteca MQTT no depende de ninguna biblioteca adicional, excepto de las siguientes:

- La biblioteca C estándar
- Una interfaz de transporte de red implementada por el cliente
- (Opcional) Una función horaria de plataforma implementada por el usuario

La biblioteca está desacoplada de los controladores de red subyacentes mediante una especificación sencilla de interfaz de transporte de envío y recepción. El autor de la aplicación puede seleccionar una interfaz de transporte existente o implementar la suya propia según convenga para su aplicación.

La biblioteca proporciona una API de alto nivel para conectarse a un agente de MQTT, suscribirse o cancelar la suscripción a un tema, publicar un mensaje en un tema y recibir los mensajes entrantes. Esta API toma como parámetro la interfaz de transporte descrita anteriormente y la utiliza para enviar y recibir mensajes desde y hacia el agente de MQTT.

La biblioteca también expone una API de serializador/deserializador de bajo nivel. Esta API se puede usar para crear una aplicación de IoT simple que consista solo en el subconjunto requerido de la funcionalidad MQTT, sin ningún otro tipo de sobrecarga. La API de serializador/deserializador se puede utilizar junto con cualquier API de capa de transporte disponible, como los sockets, para enviar y recibir mensajes desde y hacia el agente.

Cuando utilice conexiones MQTT en aplicaciones de IoT, le recomendamos que utilice una interfaz de transporte segura, como una que utilice el protocolo TLS.

Esta biblioteca MQTT no tiene dependencias de plataforma, como el subprocesamiento o la sincronización. La biblioteca tiene pruebas que muestran un uso seguro de la memoria y la ausencia de asignación de pilas, lo que la hace adecuada para microcontroladores de IoT, pero también es totalmente portátil a otras plataformas. Se puede utilizar libremente y se distribuye bajo la licencia de código abierto de MIT.

Tamaño de código de coreMQTT (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
core_mqtt.c	4,0 K	3,4 K

Tamaño de código de coreMQTT (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
core_mqtt_state.c	1,7 K	1,3 K
core_mqtt_serializer.c	2,8 K	2,2 K
Estimaciones totales	8,5 K	6,9 K

Biblioteca de agente coreMQTT

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

La biblioteca de agente coreMQTT es una API de alto nivel que añade seguridad de subprocesos a la Biblioteca coreMQTT. Permite crear una tarea de agente MQTT dedicada que gestiona una conexión MQTT en segundo plano y no necesita la intervención de otras tareas. La biblioteca proporciona equivalentes seguros para subprocesos para las API de coreMQTT, por lo que se puede utilizar en entornos con varios subprocesos.

El agente MQTT es una tarea (o subproceso de ejecución) independiente. Garantiza la seguridad de los subprocesos al ser la única tarea a la que se le permite acceder a la API de la biblioteca MQTT. Serializa el acceso aislando todas las llamadas a la API de MQTT en una sola tarea y elimina la necesidad de semáforos o cualquier otra primitiva de sincronización.

La biblioteca utiliza una cola de mensajes segura para subprocesos (u otro mecanismo de comunicación entre procesos) para serializar todas las solicitudes de llamada a las API de MQTT. La implementación de mensajería está desacoplada de la biblioteca a través de una interfaz de mensajería, que permite portar la biblioteca a otros sistemas operativos. La interfaz de mensajería se compone de funciones para enviar y recibir indicadores a las estructuras de comando del agente y funciones para asignar estos objetos de comando, lo que permite al autor de la aplicación decidir la estrategia de asignación de memoria adecuada para su aplicación.

La biblioteca está escrita en C y está diseñada para cumplir con las normas ISO C90 y MISRA C:2012. La biblioteca no depende de ninguna biblioteca adicional que no sea Biblioteca coreMQTT y la biblioteca C estándar. La biblioteca tiene pruebas que muestran un uso seguro de la memoria y la ausencia de asignación de pilas, lo que la hace adecuada para microcontroladores de IoT, pero también es totalmente portátil a otras plataformas.

Esta biblioteca se puede utilizar libremente y se distribuye bajo la licencia de código abierto de MIT.

Tamaño de código del agente coreMQTT (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
core_mqtt_agent.c	1,7 K	1,5 K
core_mqtt_agent_command_functions.c	0,3 K	0,2 K
core_mqtt.c (coreMQTT)	4,0 K	3,4 K
core_mqtt_state.c (coreMQTT)	1,7 K	1,3 K
core_mqtt_serializer.c (coreMQTT)	2,8 K	2,2 K
Estimaciones totales	10,5 K	8,6 K

Biblioteca de actualizaciones inalámbricas de AWS IoT

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

La Biblioteca de actualizaciones inalámbricas de AWS IoT le permite administrar la notificación, descarga y verificación de actualizaciones de firmware para dispositivos de FreeRTOS que utilizan HTTP o MQTT como protocolo. Mediante la biblioteca de Agente de OTA, puede separar

lógicamente las actualizaciones de firmware y la aplicación que se ejecuta en sus dispositivos. El Agente de OTA pueden compartir una conexión de red con la aplicación. Compartir una conexión de red le ofrece el potencial de ahorrar una cantidad considerable de RAM. Además, la biblioteca de Agentes de OTA le permite definir lógica específica de la aplicación para realizar pruebas, confirmar o revertir una actualización de firmware.

El Internet de las cosas (IoT) extiende la conectividad a Internet a los dispositivos integrados que tradicionalmente no estaban conectados. Estos dispositivos se pueden programar para comunicar datos utilizables a través de Internet y se pueden supervisar y controlar de forma remota. Con los avances de la tecnología, estos dispositivos integrados tradicionales están incorporando las capacidades de Internet a los espacios de consumo, industriales y empresariales a un ritmo acelerado.

Los dispositivos de IoT suelen desplegarse en grandes cantidades y, a menudo, en lugares de difícil o poco práctico acceso para un operador humano. Imagine un escenario en el que se descubre una vulnerabilidad de seguridad que puede exponer los datos. En estos escenarios, es importante actualizar los dispositivos afectados con correcciones de seguridad de forma rápida y fiable. Sin la capacidad de realizar actualizaciones OTA, también puede resultar difícil actualizar los dispositivos que están dispersos geográficamente. Hacer que un técnico actualice estos dispositivos será costoso, consumirá mucho tiempo y, a menudo, no será práctico. El tiempo necesario para actualizar estos dispositivos los expone a vulnerabilidades de seguridad durante un período más prolongado. Retirar estos dispositivos para actualizarlos también será costoso y puede causar importantes interrupciones a los consumidores debido al tiempo de inactividad.

Las actualizaciones inalámbricas permiten actualizar el firmware de los dispositivos sin tener que recurrir a costosas retiradas del mercado ni a la visita de un técnico. Este método ofrece las siguientes ventajas:

- Seguridad - La capacidad de responder rápidamente a las vulnerabilidades de seguridad y los errores de software que se descubren una vez que los dispositivos se despliegan sobre el terreno.
- Innovación - Los productos se pueden actualizar con frecuencia a medida que se desarrollan nuevas características, lo que impulsa el ciclo de innovación. Las actualizaciones pueden efectuarse rápidamente con un tiempo de inactividad mínimo en comparación con los métodos de actualización tradicionales.
- Coste - Las actualizaciones OTA pueden reducir los costes de mantenimiento de forma significativa en comparación con los métodos utilizados tradicionalmente para actualizar estos dispositivos.

Proporcionar la funcionalidad OTA requiere las siguientes consideraciones de diseño:

- Comunicación segura - Las actualizaciones deben utilizar canales de comunicación cifrados para evitar que las descargas se alteren durante el tránsito.
- Recuperación - Las actualizaciones pueden fallar debido a factores como la conectividad de red intermitente o la recepción de una actualización no válida. En estos escenarios, el dispositivo debe poder volver a un estado estable y evitar que se bloquee.
- Verificación del autor - Se debe verificar que las actualizaciones provienen de una fuente fiable, junto con otras validaciones, como comprobar la versión y la compatibilidad.

Para obtener más información acerca de cómo configurar las actualizaciones OTA con FreeRTOS, consulte Actualizaciones vía inalámbrica de FreeRTOS.

Biblioteca de actualizaciones inalámbricas de AWS IoT

La biblioteca OTA de AWS IoT le permite gestionar las notificaciones de las nuevas actualizaciones disponibles, descargarlas y realizar una verificación criptográfica de las actualizaciones del firmware. Con la biblioteca de clientes inalámbrica (OTA), puede separar de forma lógica los mecanismos de actualización del firmware de la aplicación que se ejecuta en el dispositivo. La biblioteca de clientes inalámbrica (OTA) puede compartir una conexión de red con la aplicación, lo que ahorra memoria en los dispositivos con recursos limitados. Además, la biblioteca de clientes inalámbrica (OTA) le permite definir lógica específica de la aplicación para realizar pruebas, confirmar o revertir una actualización de firmware. La biblioteca admite distintos protocolos de aplicación, como el transporte por telemetría de colas de mensajes (MQTT) y el protocolo de transferencia de hipertexto (HTTP), y ofrece varias opciones de configuración que puede ajustar con precisión al tipo y las condiciones de la red.

Las API de esta biblioteca proporcionan las siguientes funciones principales:

- Registrarse para recibir notificaciones o sondear las nuevas solicitudes de actualización que estén disponibles.
- Recibir, analizar y validar la solicitud de actualización.
- Descargar y verificar el archivo de acuerdo con la información de la solicitud de actualización.
- Realizar una autocomprobación antes de activar la actualización recibida para garantizar la validez funcional de la actualización.
- Actualizar el estado del dispositivo.

Esta biblioteca utiliza servicios de AWS para gestionar diversas funciones relacionadas con la nube, como el envío de actualizaciones de firmware, la supervisión de un gran número de dispositivos en varias regiones, la reducción del radio de impacto de las implementaciones defectuosas y la verificación de la seguridad de las actualizaciones. Esta biblioteca se puede utilizar con cualquier biblioteca MQTT o HTTP.

Las demostraciones de esta biblioteca muestran actualizaciones inalámbricas completas mediante la biblioteca de coreMQTT y los servicios de AWS en un dispositivo FreeRTOS.

Características

A continuación, se muestra la interfaz de Agente de OTA completa:

`OTA_Init`

Inicializa el motor OTA iniciando el agente OTA (“tarea OTA”) en el sistema. Solo puede existir un agente OTA.

`OTA_Shutdown`

Indica al agente de OTA que debe apagarse. El agente de OTA podrá cancelar la suscripción a todos los temas de notificaciones de trabajo de MQTT, detener las tareas de OTA en curso, si las hubiera, y borrará todos los recursos.

`OTA_GetState`

Obtiene el estado actual del Agente de OTA.

`OTA_ActivateNewImage`

Activa la imagen de firmware del microcontrolador más reciente recibida a través de OTA. (El estado del trabajo detallado ahora debe ser autodiagnóstico).

`OTA_SetImageState`

Establece el estado de validación de la imagen de firmware del microcontrolador en ejecución actualmente (prueba, aceptada o rechazada).

`OTA_GetImageState`

Obtiene el estado de la imagen de firmware del microcontrolador en ejecución actualmente (prueba, aceptada o rechazada).

`OTA_CheckForUpdate`

Solicita la siguiente actualización OTA disponible del servicio de actualización OTA.

OTA_Suspend

Suspende todas las operaciones del agente de OTA.

OTA_Resume

Reanuda las operaciones del agente de OTA.

OTA_SignalEvent

Indica un evento a la tarea del agente de OTA.

OTA_EventProcessingTask

Bucle de procesamiento de eventos del agente de OTA.

OTA_GetStatistics

Obtiene las estadísticas de los paquetes de mensajes OTA, que incluyen el número de paquetes recibidos, puestos en cola, procesados y descartados.

OTA_Err_strerror

Conversión de código de error a cadena en caso de errores de OTA.

OTA_JobParse_strerror

Convierte un código de error de análisis de trabajos de OTA en una cadena.

OTA_PalStatus_strerror

Conversión de código de estado a cadena para obtener el estado PAL de OTA.

OTA_OsStatus_strerror

Conversión de código de estado a cadena para obtener el estado OS de OTA.

Referencia de la API

Para obtener más información, consulte Actualización inalámbrica de AWS IoT: funciones.

Ejemplo de uso

Una aplicación de dispositivo típica compatible con OTA que utiliza el protocolo MQTT controla el Agente OTA mediante la siguiente secuencia de llamadas a la API.

1. Se conecta al agente de coreMQTT de AWS IoT. Para obtener más información, consulte Biblioteca de agente coreMQTT.
2. Inicializa el agente de OTA mediante una llamada a `OTA_Init`, incluidos los búferes, las interfaces OTA requeridas, el nombre del objeto y la devolución de llamada de la aplicación. La devolución de llamada implementa lógica específica de la aplicación que se ejecuta después de completar un trabajo de actualización OTA.
3. Cuando la actualización OTA se ha completado, FreeRTOS llama a la devolución de llamada de finalización del trabajo con uno de los siguientes eventos: `accepted`, `rejected` o `self test`.
4. Si se rechaza la nueva imagen de firmware (por ejemplo, debido a un error de validación), por lo general, la aplicación puede hacer caso omiso de la notificación y esperar la siguiente actualización.
5. Si la actualización es válida y se ha marcado como aceptada, llame a `OTA_ActivateNewImage` para restablecer el dispositivo e iniciar la nueva imagen de firmware.

Portabilidad

Para obtener más información sobre la función de portabilidad de OTA a su plataforma, consulte Portabilidad de la biblioteca OTA en la Guía de portabilidad de FreeRTOS.

Uso de memoria

Tamaño de código de trabajos OTA de AWS IoT (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
ota.c	8,3 K	7,5 K
ota_interface.c	0,1 K	0,1 K
ota_base64.c	0,6 K	0,6 K
ota_mqtt.c	2,4 K	2,2 K
ota_cbor.c	0,8 K	0,6 K
ota_http.c	0,3 K	0,3 K

Tamaño de código de trabajos OTA de AWS IoT (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
Estimaciones totales	12,5 K	11,3 K

Biblioteca corePKCS11

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Información general

El estándar de criptografía de clave pública 11 define una API independiente de la plataforma para administrar y usar tokens criptográficos. PKCS 11 hace referencia a la API definida por el estándar y al propio estándar. La API criptográfica de PKCS 11 abstrae propiedades get/set, de almacenamiento de claves para objetos criptográficos y semántica de la sesión. Se usa ampliamente para manipular objetos criptográficos comunes y es importante porque las funciones que especifica permiten al software de la aplicación usar, crear, modificar y eliminar objetos criptográficos sin exponerlos nunca a la memoria de la aplicación. Por ejemplo, las integraciones de referencia de AWS de FreeRTOS utilizan un pequeño subconjunto de la API PKCS 11 para acceder a la clave secreta (privada) necesaria para crear una conexión de red autenticada y protegida mediante el protocolo de seguridad de la capa de transporte (TLS) sin que la aplicación “vea” la clave.

La biblioteca corePKCS11 contiene una implementación simulada basada en software de la interfaz (API) PKCS 11 que utiliza la funcionalidad criptográfica proporcionada por Mbed TLS. El uso de una simulación de software permite un rápido desarrollo y flexibilidad, pero se espera que sustituya la simulación por una implementación específica para el almacenamiento seguro de claves que se utiliza en sus dispositivos de producción. Por lo general, los proveedores de criptoprocesadores seguros, como Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element o cualquier otro tipo de enclave de hardware seguro, distribuyen una implementación del PKCS 11 junto con el hardware. Por lo tanto, el propósito de la biblioteca simulada solo de software corePKCS11 es proporcionar una implementación PKCS 11 no específica del hardware que permita

la creación y el desarrollo rápidos de prototipos antes de cambiar a una implementación de PKCS 11 específica del criptoprocesador en los dispositivos de producción.

Solo se implementa un subconjunto del estándar PKCS 11, que se centra en las operaciones que implican claves asimétricas, la generación de números aleatorios y el hash. Los casos de uso específicos incluyen la administración de certificados y claves para la autenticación TLS y la verificación de firmas con firma de código en dispositivos integrados pequeños. Consulte el archivo `pkcs11.h` (obtenido de OASIS, el organismo de estándares) en el repositorio de código fuente de FreeRTOS. En la implementación de referencia de FreeRTOS, la interfaz de auxiliar de TLS realiza llamadas a la API PKCS 11 para realizar la autenticación de cliente de TLS durante `SOCKETS_Connect`. El flujo de trabajo de aprovisionamiento del desarrollador único también realiza llamadas a la API PKCS 11 para importar una clave privada y un certificado de cliente de TLS para la autenticación al agente MQTT de AWS IoT. Estas dos casos de uso, aprovisionamiento y autenticación de cliente de TLS, requieren la implementación de solo un pequeño subconjunto de estándares de la interfaz PKCS 11.

Características

Se utiliza el siguiente subconjunto de PKCS 11. Esta lista se encuentra aproximadamente en el orden en que se llama a las rutinas para soportar el aprovisionamiento, la autenticación de cliente de TLS y la limpieza. Para obtener descripciones detalladas de las funciones, consulte la documentación de PKCS 11 proporcionada por el comité de estándares.

API de eliminación y configuración general

- `C_Initialize`
- `C_Finalize`
- `C_GetFunctionList`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_OpenSession`
- `C_CloseSession`
- `C_Login`

API de aprovisionamiento

- `C_CreateObject CKO_PRIVATE_KEY` (para clave privada de dispositivo)

- `C_CreateObject CKO_CERTIFICATE` (para certificado de dispositivo y certificado de verificación de código)
- `C_GenerateKeyPair`
- `C_DestroyObject`

Autenticación del cliente

- `C_GetAttributeValue`
- `C_FindObjectsInit`
- `C_FindObjects`
- `C_FindObjectsFinal`
- `C_GenerateRandom`
- `C_SignInit`
- `C_Sign`
- `C_VerifyInit`
- `C_Verify`
- `C_DigestInit`
- `C_DigestUpdate`
- `C_DigestFinal`

Soporte de criptosistema asimétrico

La implementación de referencia de FreeRTOS utiliza RSA de 2048 bits de PKCS 11 (solo firma) y ECDSA con la curva NIST P-256. Las siguientes instrucciones se describe cómo crear un AWS IoT objeto en función de un P-256 certificado de cliente.

Asegúrese de que está utilizando las siguientes (o más reciente) versiones del AWS CLI y OpenSSL:

```
aws --version
aws-cli/1.11.176 Python/2.7.9 Windows/8 botocore/1.7.34

openssl version
OpenSSL 1.0.2g  1 Mar 2016
```

En el siguiente procedimiento se supone que ha utilizado el comando `aws configure` para configurar la AWS CLI. Para obtener más información, consulte Configuración rápida con `aws configure` en la Guía del usuario de AWS Command Line Interface.

Para crear un objeto de AWS IoT basado en certificado de cliente P-256

1. Cree un objeto de AWS IoT.

```
aws iot create-thing --thing-name thing-name
```

2. Utilice OpenSSL para crear una clave P-256.

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out thing-name.key
```

3. Cree una solicitud de inscripción de certificado firmada con la clave que creó en el paso 2.

```
openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
```

4. Envíe la solicitud de inscripción del certificado a AWS IoT.

```
aws iot create-certificate-from-csr  \
  --certificate-signing-request file://thing-name.req --set-as-active  \
  --certificate-pem-outfile thing-name.crt
```

5. Asocie el certificado (al que la salida de ARN hace referencia en el comando anterior) al objeto.

```
aws iot attach-thing-principal --thing-name thing-name \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

6. Cree una política. (Esta política es demasiado permisiva. Debe usarse únicamente con fines de desarrollo).

```
aws iot create-policy --policy-name FullControl --policy-document file://
policy.json
```

   A continuación se muestra una lista del archivo policy.json especificado en el comando `create-policy`. Puede omitir la acción `greengrass:*` si no desea ejecutar la demostración de FreeRTOS para la detección y conectividad Greengrass.

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

7. Asocie la entidad de seguridad (certificado) y la política al objeto.

```
aws iot attach-principal-policy --policy-name FullControl \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

Ahora, siga los pasos que se indican en la sección Introducción a AWS IoT de esta guía. No olvide copiar el certificado y la clave privada que creó en su archivo `aws_clientcredential_keys.h`. Copie el nombre del objeto a `aws_clientcredential.h`.

> **Note**
>
> El certificado y la clave privada se incluyen en el código solo para fines de demostración. Las aplicaciones de producción deben almacenar estos archivos en un lugar seguro.

Portabilidad

Para obtener información acerca de la portabilidad de la biblioteca corePKCS11 a su plataforma, consulte Portabilidad de la biblioteca corePKCS11 en la Guía de portabilidad de FreeRTOS.

Uso de memoria

Tamaño de código de corePKCS11 (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
core_pkcs11.c	0,8 K	0,8 K
core_pki_utils.c	0,5 K	0,3 K
core_pkcs11_mbedtls.c	8,9 K	7,5 K
Estimaciones totales	10,2 K	8,6 K

Biblioteca de sockets seguros

> **Important**
>
> Esta biblioteca está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Información general

Puede utilizar la biblioteca de sockets seguros de FreeRTOS para crear aplicaciones integradas que se comunican de forma segura. La biblioteca se ha diseñado para facilitar la incorporación de desarrolladores de software de diversos antecedentes de programación de red.

La biblioteca de sockets seguros de FreeRTOS se basa en la interfaz de sockets de Berkeley, con una opción de comunicación segura adicional mediante el protocolo TLS. Para obtener más información acerca de las diferencias entre la biblioteca de sockets seguros de FreeRTOS y la interfaz de sockets de Berkeley, consulte `SOCKETS_SetSockOpt` en la Referencia de la API de sockets seguros.

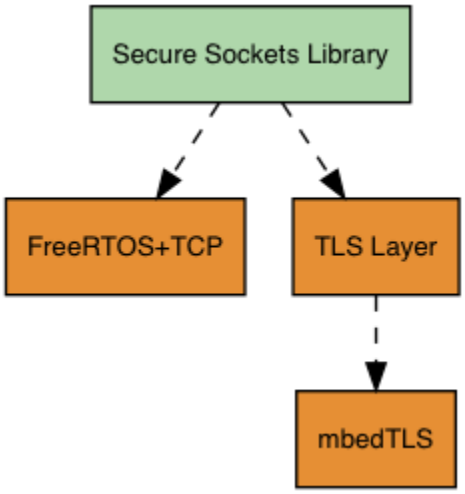
> **Note**
>
> Actualmente, solo las API de cliente, además de una implementación de IP ligera (lwIP) de la API `Bind` del lado del servidor, son compatibles con sockets seguros de FreeRTOS.

Dependencias y requisitos

La biblioteca de sockets seguros de FreeRTOS depende de una pila TCP/IP y de una implementación de TLS. Las portabilidades para FreeRTOS cumplen estas dependencias en una de las tres formas:

- Una implementación personalizada de TCP/IP y TLS
- Una implementación personalizada de TCP/IP y la capa TLS de FreeRTOS con mbedTLS
- FreeRTOS+TCP y la capa TLS de FreeRTOS con mbedTLS

El diagrama de dependencias siguiente muestra la implementación de referencia incluida con la biblioteca de sockets seguros de FreeRTOS. Esta implementación de referencia admite TLS y TCP/IP a través de Ethernet y Wi-Fi con FreeRTOS+TCP y mbedTLS como dependencias. Para obtener más información acerca de la capa TLS de FreeRTOS, consulte Transport Layer Security.



Características

Las características de la biblioteca de sockets seguros de FreeRTOS incluyen:

- Una interfaz estándar basada en sockets Berkeley
- API seguras para subprocesos para el envío y recepción de datos

- asy-to-enable TLS

Solución de problemas

Códigos de error

Los códigos de error que la biblioteca de sockets seguros de FreeRTOS devuelve son valores negativos. Para obtener más información acerca de cada código de error, consulte Códigos de error de sockets seguros en la Referencia de la API de sockets seguros.

> **Note**
>
> Si la API de sockets seguros de FreeRTOS devuelve un código de error, Biblioteca coreMQTT, que depende de la biblioteca de sockets seguros de FreeRTOS devuelve el código de error `AWS_IOT_MQTT_SEND_ERROR`.

Developer Support

La biblioteca de sockets seguros de FreeRTOS incluye dos macros auxiliares para gestionar las direcciones IP:

`SOCKETS_inet_addr_quick`

Esta macro convierte una dirección IP que se expresa como cuatro octetos numéricos independientes en una dirección IP que se expresa como un número de 32 bits en orden de bytes de red.

`SOCKETS_inet_ntoa`

Esta macro convierte una dirección IP que se expresa como un número de 32 bits en orden de bytes de red en una cadena en notación decimal con puntos.

Restricciones de uso

Solo se admiten sockets de TCP mediante la biblioteca de sockets seguros de FreeRTOS. No se admiten los sockets de UDP.

La biblioteca de sockets seguros de FreeRTOS no admite las API de servidor, excepto la implementación de IP ligera (lwIP) de la API `Bind` del lado del servidor. Se admiten las API de cliente.

Inicialización

Para utilizar la biblioteca de sockets seguros de FreeRTOS, tiene que inicializar la biblioteca y sus dependencias. Para inicializar la biblioteca de sockets seguros, utilice el siguiente código en su aplicación:

```
BaseType_t xResult = pdPASS;
xResult = SOCKETS_Init();
```

Las bibliotecas dependientes deben inicializarse por separado. Por ejemplo, si FreeRTOS+TCP es una dependencia, necesita invocar también `FreeRTOS_IPInit` en su aplicación.

Referencia de la API

Para obtener una referencia completa de la API, consulte la Referencia de la API de sockets seguros.

Ejemplo de uso

Con el siguiente código se conecta un cliente a un servidor.

```
#include "aws_secure_sockets.h"

#define configSERVER_ADDR0                    127
#define configSERVER_ADDR1                    0
#define configSERVER_ADDR2                    0
#define configSERVER_ADDR3                    1
#define configCLIENT_PORT                     443

/* Rx and Tx timeouts are used to ensure the sockets do not wait too long for
 * missing data. */
static const TickType_t xReceiveTimeOut = pdMS_TO_TICKS( 2000 );
static const TickType_t xSendTimeOut = pdMS_TO_TICKS( 2000 );

/* PEM-encoded server certificate */
/* The certificate used below is one of the Amazon Root CAs.\
Change this to the certificate of your choice. */
static const char cTlsECHO_SERVER_CERTIFICATE_PEM[] =
"-----BEGIN CERTIFICATE-----\n"
"MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQQDAjA5\n"
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"
"Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"
"A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"
```

```
"Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG8lKl\n"
"ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVOho6AF2hiRVd9RFgdszflZwjrZt6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkjOPQQDAgNJADBGAiEA4IWSoxe3jfkr\n"
"BqWTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyzlvnrxir4tiz+OpAUFteM\n"
"YyRIHN8wfdVoOw==\n"
"-----END CERTIFICATE-----\n";

static const uint32_t ulTlsECHO_SERVER_CERTIFICATE_LENGTH =
 sizeof( cTlsECHO_SERVER_CERTIFICATE_PEM );

void vConnectToServerWithSecureSocket( void )
{
    Socket_t xSocket;
    SocketsSockaddr_t xEchoServerAddress;
    BaseType_t xTransmitted, lStringLength;

    xEchoServerAddress.usPort = SOCKETS_htons( configCLIENT_PORT );
    xEchoServerAddress.ulAddress = SOCKETS_inet_addr_quick( configSERVER_ADDR0,
                                                            configSERVER_ADDR1,
                                                            configSERVER_ADDR2,
                                                            configSERVER_ADDR3 );

    /* Create a TCP socket. */
    xSocket = SOCKETS_Socket( SOCKETS_AF_INET, SOCKETS_SOCK_STREAM,
 SOCKETS_IPPROTO_TCP );
    configASSERT( xSocket != SOCKETS_INVALID_SOCKET );

    /* Set a timeout so a missing reply does not cause the task to block indefinitely.
 */
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_RCVTIMEO, &xReceiveTimeOut,
 sizeof( xReceiveTimeOut ) );
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_SNDTIMEO, &xSendTimeOut,
 sizeof( xSendTimeOut ) );

    /* Set the socket to use TLS. */
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_REQUIRE_TLS, NULL, ( size_t ) 0 );
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE,
 cTlsECHO_SERVER_CERTIFICATE_PEM, ulTlsECHO_SERVER_CERTIFICATE_LENGTH );

    if( SOCKETS_Connect( xSocket, &xEchoServerAddress, sizeof( xEchoServerAddress ) )
 == 0 )
    {
        /* Send the string to the socket. */
```

```
        xTransmitted = SOCKETS_Send( xSocket,                          /* The socket
 receiving. */
                                     ( void * )"some message",         /* The data being
 sent. */
                                     12,                               /* The length of
 the data being sent. */
                                     0 );                              /* No flags. */

        if( xTransmitted < 0 )
        {
            /* Error while sending data */
            return;
        }

        SOCKETS_Shutdown( xSocket, SOCKETS_SHUT_RDWR );
    }
    else
    {
        //failed to connect to server
    }

    SOCKETS_Close( xSocket );
}
```

Para ver un ejemplo completo, consulte Demostración de cliente de eco de sockets seguros.

Portabilidad

Los sockets seguros de FreeRTOS dependen de una pila TCP/IP y de una implementación de TLS. En función de la pila, para realizar la portabilidad de la biblioteca de sockets seguros, es posible que tenga que trasladar algunos de los siguientes:

- La pila TCP/IP FreeRTOS+TCP
- Con la Biblioteca corePKCS11
- Con la Transport Layer Security

Para obtener más información sobre la portabilidad, consulte Portabilidad de la biblioteca de sockets seguros en la Guía de portabilidad de FreeRTOS.

Biblioteca de sombras de dispositivos de AWS IoT

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

Puede utilizar la biblioteca de sombra de dispositivo de AWS IoT para almacenar y recuperar el estado actual (la sombra) de todos los dispositivos registrados. La sombra del dispositivo es una representación virtual y persistente de su dispositivo con la que puede interactuar en sus aplicaciones web aunque el dispositivo esté desconectado. El estado del dispositivo se captura como su sombra en un documento JSON. Puede enviar comandos al servicio de sombra de dispositivo de AWS IoT a través de MQTT o HTTP para consultar el último estado conocido del dispositivo o para cambiarlo. La sombra de cada dispositivo se identifica de forma única con el nombre del objeto correspondiente, una representación de un dispositivo o entidad lógica específicos en la nube de AWS. Para obtener más información, consulte Administración de dispositivos con AWS IoT. Se pueden encontrar más detalles sobre las sombras en la documentación de AWS IoT.

La biblioteca de sombra de dispositivo de AWS IoT no depende de ninguna biblioteca adicional que no sea la biblioteca C estándar. Tampoco tiene dependencias de plataforma, como el subprocesamiento o la sincronización. Se puede usar con cualquier biblioteca MQTT y JSON.

Esta biblioteca se puede utilizar libremente y se distribuye bajo la licencia de código abierto de MIT.

Tamaño de código de sombra de dispositivo de AWS IoT (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
shadow.c	1,2 K	0,9 K
Estimaciones totales	1,2 K	0,9 K

Biblioteca de trabajos de AWS IoT

> **Note**
>
> Es posible que el contenido de esta página no esté actualizado. Consulte la página de la biblioteca de FreeRTOS.org para obtener la última actualización.

Introducción

Trabajos de AWS IoT es un servicio que notifica a uno o más dispositivos conectados un trabajo pendiente. Puede usar un trabajo para administrar su flota de dispositivos, actualizar el firmware y los certificados de seguridad de sus dispositivos o realizar tareas administrativas, como reiniciar los dispositivos y realizar diagnósticos. Para obtener más información, consulte Empleos en la Guía para desarrolladores de AWS IoT. Las interacciones con Trabajos de AWS IoT utilizan MQTT, un protocolo ligero de publicación y suscripción. Esta biblioteca proporciona una API para componer y reconocer las cadenas de temas MQTT que utiliza el servicio de trabajos de AWS IoT.

La biblioteca de trabajos de AWS IoT está escrita en C y está diseñada para cumplir con las normas ISO C90 y MISRA C:2012. La biblioteca no depende de ninguna biblioteca adicional que no sea la biblioteca C estándar. Se puede usar con cualquier biblioteca MQTT y JSON. La biblioteca tiene pruebas que muestran un uso seguro de la memoria y la ausencia de asignación de pilas, lo que la hace adecuada para microcontroladores de IoT, pero también es totalmente portátil a otras plataformas.

Esta biblioteca se puede utilizar libremente y se distribuye bajo la licencia de código abierto de MIT.

Tamaño de código de trabajos de AWS IoT (ejemplo generado con GCC para ARM Cortex-M)		
Archivos	Con optimización -O1	Con optimización -Os
jobs.c	1,9 K	1,6 K
Estimaciones totales	1,9 K	1,6 K

Transport Layer Security

> **Important**
>
> Esta biblioteca está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

La interfaz de seguridad de la capa de transporte (TLS) de FreeRTOS es un contenedor ligero y opcional usado para abstraer detalles de implementación criptográfica de la interfaz de la capa de sockets seguros (SSL) sobre la pila del protocolo. El objetivo de la interfaz de TLS es que la biblioteca de criptografía de software actual, mbed TLS, sea fácil de sustituir por una implementación alternativa para la negociación de protocolos de TLS y primitivos criptográficos. La interfaz TLS se puede cambiar sin que sea necesario realizar cambios en la interfaz SSL. Consulte `iot_tls.h` en el repositorio de códigos fuente de FreeRTOS.

La interfaz TLS es opcional porque puede elegir establecer una conexión directamente desde SSL a una biblioteca de criptografía. La interfaz no se utiliza para soluciones de MCU que incluyen una implementación de descarga "full stack" de transporte de red y TLS.

Para obtener más información sobre la portabilidad de la interfaz TLS, consulte Portabilidad de la biblioteca de TLS en la Guía de portabilidad de FreeRTOS.

Biblioteca wifi

> **Important**
>
> Esta biblioteca está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Información general

La biblioteca Wi-Fi de FreeRTOS abstrae las implementaciones Wi-Fi específicas de puerto en una API común que simplifica el desarrollo de aplicaciones y la portabilidad para todas las placas

calificadas para FreeRTOS con capacidades Wi-Fi. Con esta API común, las aplicaciones pueden comunicarse con su pila inalámbrica de bajo nivel a través de una interfaz común.

Dependencias y requisitos

La biblioteca Wi-Fi de FreeRTOS requiere el núcleo FreeRTOS+TCP.

Características

La biblioteca Wi-Fi incluye las siguientes características:

- Compatibilidad para la autenticación WEP, WPA, WPA2 y WPA3
- Búsqueda de punto de acceso
- Administración de energía
- Perfiles de red

Para obtener más información sobre las características de la biblioteca Wi-Fi, consulte lo siguiente:

Modos Wi-Fi

Los dispositivos Wi-Fi pueden encontrarse en uno de tres modos: Estación, Punto de acceso, o P2P. Para obtener el modo actual de un dispositivo wifi, llame a `WIFI_GetMode`. Puede establecer el modo Wi-Fi del dispositivo llamando a `WIFI_SetMode`. El cambio de modos llamando a `WIFI_SetMode` desconecta el dispositivo, si ya está conectado a una red.

Modo de estación

Establezca su dispositivo en el modo Estación para conectar la placa a un punto de acceso existente.

Modo de punto de acceso (AP)

Establezca su dispositivo en modo AP para convertir al dispositivo en un punto de acceso para conectar a otros dispositivos. Cuando el dispositivo está en modo AP, puede conectar otro dispositivo a su dispositivo FreeRTOS y configurar las nuevas credenciales Wi-Fi. Para configurar el modo AP, llame a `WIFI_ConfigureAP`. Para poner el dispositivo en modo AP, llame a `WIFI_StartAP`. Para desactivar el modo AP, llame a `WIFI_StopAP`.

> **Note**
>
> Las bibliotecas de FreeRTOS no proporcionan aprovisionamiento Wi-Fi en modo AP. Debe proporcionar la funcionalidad adicional, incluidas las capacidades del servidor DHCP y HTTP, para lograr la compatibilidad completa del modo AP.

Modo P2P

Establezca su dispositivo al modo P2P para permitir que varios dispositivos se conecten entre sí directamente, sin punto de acceso.

Seguridad

La API de Wi-Fi admite los tipos de seguridad WEP, WPA, WPA2 y WPA3. Si el dispositivo se encuentra en modo Estación, debe especificar el tipo de seguridad de la red cuando llame a la función `WIFI_ConnectAP`. Si el dispositivo está en modo AP, es posible configurar el dispositivo para que utilice cualquiera de los tipos de seguridad admitidos:

- `eWiFiSecurityOpen`
- `eWiFiSecurityWEP`
- `eWiFiSecurityWPA`
- `eWiFiSecurityWPA2`
- `eWiFiSecurityWPA3`

Búsqueda y conexión

Para buscar puntos de acceso cercanos, establezca su dispositivo en modo Estación y llame a la función `WIFI_Scan`. Si encuentra una red deseada en la búsqueda, puede conectarse a la red llamando a `WIFI_ConnectAP` y proporcionando las credenciales de red. Para desconectar el dispositivo wifi de la red, llame a `WIFI_Disconnect`. Para obtener más información acerca de la búsqueda y conexión, consulte Ejemplo de uso y Referencia de la API.

Administración de energía

Los distintos dispositivos wifi tienen requisitos de energía diferentes en función de la aplicación y las fuentes de energía disponibles. Un dispositivo podría estar siempre encendido para reducir

la latencia o podría estar conectado de forma intermitente y cambiar a un modo de bajo consumo cuando no se necesita conexión wifi. La API de interfaz admite varios modos de administración de energía, como siempre, bajo consumo y modo normal. Puede establecer el modo de energía para un dispositivo con la función `WIFI_SetPMMode`. Para obtener el modo de energía actual de un dispositivo, llame a la función `WIFI_GetPMMode`.

Perfiles de red

La biblioteca Wi-Fi le permite guardar perfiles de red en la memoria no volátil de los dispositivos. Esto permite guardar ajustes de la red que se pueden recuperar cuando el dispositivo se vuelve a conectar a una red wifi, eliminando la necesidad de aprovisionar dispositivos de nuevo después de que se hayan conectado a una red. `WIFI_NetworkAdd` añade un perfil de red. `WIFI_NetworkGet` recupera un perfil de red. `WIFI_NetworkDel` elimina un perfil de red. El número de perfiles que puede guardar depende de la plataforma.

Configuración

Para utilizar la biblioteca Wi-Fi, es necesario definir varios identificadores en un archivo de configuración. Para obtener información sobre estos identificadores, consulte la Referencia de la API.

> **Note**
>
> La biblioteca no incluye el archivo de configuración requerido. Debe crear uno. Al crear su archivo de configuración, asegúrese de incluir los identificadores de configuración específicos de placa que requiere su placa.

Inicialización

Antes de utilizar la biblioteca Wi-Fi, tiene que inicializar algunos componentes específicos de placa, además de los componentes de FreeRTOS. Con el archivo `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/application_code/main.c` como plantilla para la inicialización, haga lo siguiente:

1. Elimine el ejemplo de lógica de conexión Wi-Fi en `main.c` si la aplicación controla las conexiones Wi-Fi. Sustituya la siguiente llamada a la función `DEMO_RUNNER_RunDemos()`:

```
if( SYSTEM_Init() == pdPASS )
    {
    ...
        DEMO_RUNNER_RunDemos();
```

```
    ...
        }
```

Con una llamada a su propia aplicación:

```
if( SYSTEM_Init() == pdPASS )
    {
    ...
        // This function should create any tasks
        // that your application requires to run.
        YOUR_APP_FUNCTION();
    ...
        }
```

2. Llame a `WIFI_On()` para inicializar y activar su chip Wi-Fi.

> **Note**
>
> Algunas placas podrían necesitar inicialización de hardware adicional.

3. Pase la estructura `WIFINetworkParams_t` configurada a `WIFI_ConnectAP()` para conectar su placa a una red Wi-Fi disponible. Para obtener más información sobre la estructura `WIFINetworkParams_t`, consulte Ejemplo de uso y Referencia de la API.

Referencia de la API

Para obtener una referencia de la API completa, consulte la Referencia de la API Wi-Fi.

Ejemplo de uso

Conexión a un AP conocido

```
#define clientcredentialWIFI_SSID    "MyNetwork"
#define clientcredentialWIFI_PASSWORD   "hunter2"

WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

xWifiStatus = WIFI_On(); // Turn on Wi-Fi module

// Check that Wi-Fi initialization was successful
```

```
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi library initialized.\n") );
}
else
{
    configPRINT( ( "WiFi library failed to initialize.\n" ) );
    // Handle module init failure
}

/* Setup parameters. */
xNetworkParams.pcSSID = clientcredentialWIFI_SSID;
xNetworkParams.ucSSIDLength = sizeof( clientcredentialWIFI_SSID );
xNetworkParams.pcPassword = clientcredentialWIFI_PASSWORD;
xNetworkParams.ucPasswordLength = sizeof( clientcredentialWIFI_PASSWORD );
xNetworkParams.xSecurity = eWiFiSecurityWPA2;

// Connect!
xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );

if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi Connected to AP.\n" ) );
    // IP Stack will receive a network-up event on success
}
else
{
    configPRINT( ( "WiFi failed to connect to AP.\n" ) );
    // Handle connection failure
}
```

Búsqueda de AP cercanos

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

configPRINT( ("Turning on wifi...\n") );
xWifiStatus = WIFI_On();

configPRINT( ("Checking status...\n") );
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ("WiFi module initialized.\n") );
```

```
}
else
{
    configPRINTF( ("WiFi module failed to initialize.\n" ) );
    // Handle module init failure
}

WIFI_SetMode(eWiFiModeStation);

/* Some boards might require additional initialization steps to use the Wi-Fi library.
 */

while (1)
{
    configPRINT( ("Starting scan\n") );
    const uint8_t ucNumNetworks = 12; //Get 12 scan results
    WIFIScanResult_t xScanResults[ ucNumNetworks ];
    xWifiStatus = WIFI_Scan( xScanResults, ucNumNetworks ); // Initiate scan

    configPRINT( ("Scan started\n") );

    // For each scan result, print out the SSID and RSSI
    if ( xWifiStatus == eWiFiSuccess )
    {
        configPRINT( ("Scan success\n") );
        for ( uint8_t i=0; i<ucNumNetworks; i++ )
        {
            configPRINTF( ("%s : %d \n", xScanResults[i].cSSID,
 xScanResults[i].cRSSI) );
        }
    } else {
        configPRINTF( ("Scan failed, status code: %d\n", (int)xWifiStatus) );
    }

    vTaskDelay(200);
}
```

Portabilidad

La implementación de `iot_wifi.c` debe implementar las funciones definidas en `iot_wifi.h`. Como mínimo, la implementación debe devolver `eWiFiNotSupported` para cualquier función no esencial o no admitida.

Para obtener más información sobre la portabilidad de la biblioteca Wi-Fi, consulte Portabilidad de la biblioteca Wi-Fi en la Guía de portabilidad de FreeRTOS.

Demostraciones de FreRTOS

FreeRTOS incluye algunas aplicaciones de demostración en la carpeta `demos`, en el directorio principal de FreeRTOS. Todos los ejemplos que puede ejecutar FreeRTOS aparecen en la carpeta `common`, en `demos`. También hay una carpeta para cada plataforma calificada para FreeRTOS en la carpeta `demos`.

Antes de probar la aplicaciones de demostración, le recomendamos que complete el tutorial Introducción a FreeRTOS. Muestra cómo configurar y ejecutar la demostración de agente coreMQTT.

Ejecución de demostraciones de FreeRTOS

En los siguientes temas se muestra cómo configurar y ejecutar las demostraciones de FreeRTOS:

- Aplicaciones de demostración de Bluetooth de bajo consumo
- Cargador de arranque de demostración para la placa de desarrollo Curiosity PIC32MZEF de Microchip
- Demostración de AWS IoT Device Defender
- Aplicación de demostración de detección de AWS IoT Greengrass V1
- AWS IoT Greengrass V2
- Demostraciones de coreHTTP
- Demostración de la biblioteca de trabajos de AWS IoT
- Demostraciones de coreMQTT
- Aplicación de demostración de actualizaciones transparentes
- Demostración de cliente de eco de sockets seguros
- Aplicación de demostración de sombra de dispositivos AWS IoT

La función `DEMO_RUNNER_RunDemos`, que se encuentra en el archivo *`freertos`*`/demos/demo_runner/iot_demo_runner.c`, inicializa un subproceso independiente en el que se ejecuta una única aplicación de demostración. De forma predeterminada, `DEMO_RUNNER_RunDemos` solo llama e inicia la demostración de agente coreMQTT. En función de la configuración que haya seleccionado al descargar FreeRTOS y dependiendo del lugar desde donde haya descargado FreeRTOS, las otras funciones de ejecutor de ejemplo podrían comenzar de forma predeterminada.

Para habilitar una aplicación de demostración, abra el archivo *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h y defina la demostración que desee ejecutar.

> **Note**
>
> Tenga en cuenta que no todas las combinaciones de ejemplos funcionan conjuntamente. En función de la combinación, el software podría no ejecutarse en el destino seleccionado debido a limitaciones de memoria. Le recomendamos que ejecute una demostración a la vez.

Configuración de las demostraciones

Las demostraciones se han configurado para que pueda empezar rápidamente. Es posible que desee cambiar algunas de las configuraciones de su proyecto para crear una versión que se ejecuta en su plataforma. Encontrará los archivos de configuración en vendors/*vendor*/boards/*board*/aws_demos/config_files.

Aplicaciones de demostración de Bluetooth de bajo consumo

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Información general

Bluetooth de bajo consumo de FreeRTOS incluye tres aplicaciones de demostración:

- Demostración de MQTT a través de Bluetooth de bajo consumo

  Esta aplicación muestra cómo utilizar el servicio MQTT a través de Bluetooth de bajo consumo.
- Demostración de Aprovisionamiento Wi-Fi

  Esta aplicación muestra cómo utilizar el servicio de Aprovisionamiento Wi-FI de Bluetooth de bajo consumo.

- Demostración de Servidor de atributos genéricos

  Esta aplicación muestra cómo utilizar las API de middleware de Bluetooth de bajo consumo de FreeRTOS para crear un servidor de GATT sencillo.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Requisitos previos

Para seguir estas demostraciones, necesita un microcontrolador con capacidades Bluetooth de bajo consumo. También necesita el SDK para iOS para dispositivos Bluetooth de FreeRTOS o el SDK para Android para dispositivos Bluetooth de FreeRTOS.

Configuración de AWS IoT y Amazon Cognito para Bluetooth de bajo consumo de FreeRTOS

Para conectar sus dispositivos a AWS IoT a través de MQTT, debe configurar AWS IoT y Amazon Cognito.

Para configurar AWS IoT

1. Configure una cuenta de AWS en https://aws.amazon.com/.
2. Abra la consola de AWS IoT y en el panel de navegación, elija Manage (Administrar) y después Things (Objetos).
3. Elija Create (Crear) y después Create a single thing (Crear un solo objeto).
4. Escriba un nombre para el dispositivo y después elija Next (Siguiente).
5. Si va a conectar su microcontrolador a la nube a través de un dispositivo móvil, elija Create thing without certificate (Crear objeto sin certificado). Debido a que los SDK para móviles utilizan Amazon Cognito para autenticar los dispositivos, no es necesario crear un certificado de dispositivo para las demostraciones que utilizan Bluetooth de bajo consumo.

   Si va a conectar su microcontrolador a la nube directamente a través de Wi-Fi, seleccione Create certificate (Crear certificado), elija Activate (Activar) y, a continuación, descargue el certificado del objeto, una clave pública y una clave privada.

6. Elija el objeto que acaba de crear en la lista de objetos registrados y, a continuación, elija Interact (Interactuar) en la página del objeto. Tome nota del punto de enlace de la API REST de AWS IoT.

Para obtener más información sobre la configuración, consulte Introducción a AWS IoT.

Para crear un grupo de usuarios de Amazon Cognito

1. Abra la consola de Amazon Cognito y elija Administrar los grupos de usuarios.
2. Elija Create a user pool.
3. Asigne un nombre al grupo de usuarios y, a continuación, elija Review defaults (Revisar opciones predeterminadas).
4. En el panel de navegación, elija App clients (Clientes de aplicación) y después elija Add an app client (Añadir un cliente de aplicación).
5. Escriba un nombre para el cliente de aplicación y, a continuación, seleccione Create app client (Crear cliente de aplicación).
6. En el panel de navegación, seleccione Review (Revisar) y, a continuación, seleccione Create pool (Crear grupo).

   Anote el ID de grupo que aparece en la página General Settings (Configuración general) del grupo de usuarios.
7. En el panel de navegación, elija App clients (Clientes de aplicación) y después elija Show details (Mostrar detalles). Anote el ID y el secreto del cliente de aplicación.

Para crear un grupo de identidades de Amazon Cognito

1. Abra la consola de Amazon Cognito y elija Administrar los grupos de identidades.
2. Escriba un nombre para el grupo de identidades.
3. Expanda Authentication providers (Proveedores de autenticación), elija la pestaña Cognito y después escriba el ID del grupo de usuarios y el ID del cliente de aplicación.
4. Elija Create Pool (Crear grupo).
5. Expanda View Details (Ver detalles) y anote los dos nombres de roles de IAM. Elija Permitir para crear los roles de IAM para que las identidades autenticadas y no autenticadas tengan acceso a Amazon Cognito.

6. Elija Edit identity pool (Editar grupo de identidades). Anote el ID del grupo de identidades. Debe tener el formato `us-west-2:12345678-1234-1234-1234-123456789012`.

Para obtener más información sobre cómo configurar Amazon Cognito, consulte Introducción a Amazon Cognito.

Para crear y asociar una política de IAM a la identidad autenticada

1. Abra la consola de IAM y, en el panel de navegación, elija Roles.
2. Busque y seleccione el rol de la identidad autenticada, elija Attach policies (Asociar políticas) y, a continuación, elija Add inline policy (Añadir política insertada).
3. Elija la pestaña JSON y pegue el siguiente código JSON:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:AttachPolicy",
            "iot:AttachPrincipalPolicy",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "iot:DeleteThingShadow"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

4. Elija Review policy (Revisar política), escriba un nombre para la política y después elija Create policy (Crear política).

Mantenga a mano su información de AWS IoT y Amazon Cognito. Necesita el punto de enlace y los ID para autenticar su aplicación móvil con la nube de AWS.

Configuración del entorno de FreeRTOS para Bluetooth de bajo consumo

Para configurar su entorno, necesita descargar FreeRTOS con la Biblioteca de Bluetooth de bajo consumo en su microcontrolador y descargar y configurar el SDK para móviles para dispositivos Bluetooth de FreeRTOS en su dispositivo móvil.

Para configurar el entorno del microcontrolador con Bluetooth de bajo consumo de FreeRTOS

1. Descargue o clone FreeRTOS desde GitHub. Consulte el archivo README.md para obtener instrucciones.
2. Configure FreeRTOS en su microcontrolador.

   Para obtener información acerca de cómo comenzar a utilizar FreeRTOS en un microcontrolador calificado para FreeRTOS, consulte la guía de su placa en Introducción a FreeRTOS.

   > **Note**
   >
   > Puede ejecutar las demostraciones en cualquier microcontrolador habilitado para Bluetooth de bajo consumo con FreeRTOS y bibliotecas de Bluetooth de bajo consumo de FreeRTOS portadas. En la actualidad, el proyecto de demostración MQTT a través de Bluetooth de bajo consumo de FreeRTOS se transfiere totalmente a los siguientes dispositivos habilitados para Bluetooth de bajo consumo:
   >
   > - Espressif ESP32-DevKitC y ESP-WROVER-KIT
   > - Nordic nRF52840-DK

Componentes comunes

Las aplicaciones de demostración de FreeRTOS tienen dos componentes comunes:

- Administrador de red
- Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo

Administrador de red

El Administrador de red administra la conexión de red del microcontrolador. Se encuentra en su directorio de FreeRTOS en `demos/network_manager/aws_iot_network_manager.c`. Si el administrador de red está habilitado para Wi-Fi y Bluetooth de bajo consumo, las demostraciones comienzan con Bluetooth de bajo consumo de forma predeterminada. Si se interrumpe la conexión de Bluetooth de bajo consumo y su placa tiene habilitada Wi-Fi, el administrador de red cambia a una conexión Wi-Fi disponible para impedir que se desconecte de la red.

Para habilitar un tipo de conexión de red con el administrador de red, agregue el tipo de conexión de red al parámetro `configENABLED_NETWORKS` de `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h` (donde *`vendor`* es el nombre del proveedor y *`board`* es el nombre de la placa que utiliza para ejecutar las demostraciones).

Por ejemplo, si tiene Bluetooth de bajo consumo y Wi-Fi habilitados, la línea que comienza con `#define configENABLED_NETWORKS` en `aws_iot_network_config.h` lee como se indica a continuación:

```
#define  configENABLED_NETWORKS  ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

Para obtener una lista de los tipos de conexión de red que se admiten actualmente, consulte las líneas que comienzan por `#define AWSIOT_NETWORK_TYPE` en `aws_iot_network.h`.

Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS

La aplicación de demostración del SDK para móviles de Bluetooth de bajo consumo de FreeRTOS se encuentra en GitHub, en el SDK para Android para dispositivos Bluetooth de bajo consumo de FreeRTOS en `amazon-freertos-ble-android-sdk/app` y el SDK para iOS para dispositivos Bluetooth de bajo consumo de FreeRTOS en `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo`. En este ejemplo, utilizamos capturas de pantalla de la versión iOS de la aplicación de demostración móvil.

> **Note**
>
> Si está utilizando un dispositivo iOS, necesita Xcode para crear la aplicación móvil de demostración. Si está utilizando un dispositivo Android, puede utilizar Android Studio para crear la aplicación móvil de demostración.

Para configurar la aplicación de demostración del SDK de iOS

Cuando defina las variables de configuración, utilice el formato del los valores de marcador de posición proporcionados en los archivos de configuración.

1. Confirme que ha instalado el SDK para iOS para dispositivos Bluetooth de FreeRTOS.
2. Ejecute el siguiente comando desde `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/`:

```
$ pod install
```

3. Abra el proyecto `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace` con Xcode y cambie la cuenta del desarrollador de la firma por su cuenta.
4. Cree una política de AWS IoT en su región (si aún no lo ha hecho).

> **Note**
>
> Esta política es diferente de la política de IAM creada para la identidad autenticada de Amazon Cognito.

   a. Abra la consola de AWS IoT.
   b. En el panel de navegación, elija Secure (Seguridad), elija Policies (Políticas) y, a continuación, elija Create (Crear). Especifique un nombre que identifique la política. En la sección Añadir declaraciones, elija Modo avanzado. Copie y pegue la siguiente política JSON en la ventana del editor de políticas. Sustituya *aws-region* y *aws-account* por su región e ID de cuenta de AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource":"arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Subscribe",
             "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Receive",
             "Resource": "arn:aws:iot:region:account-id:*"
        }
    ]
}
```

c. Seleccione Crear.

5. Abra `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift` y redefina las siguientes variables:

   - `region`: su región de AWS.
   - `iotPolicyName`: el nombre de la política de AWS IoT.
   - `mqttCustomTopic`: el tema de MQTT en el que desea publicar.

6. Abrir `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json`.

   En `CognitoIdentity`, redefina las siguientes variables:

   - `PoolId`: su ID de grupo de identidades de Amazon Cognito.
   - `Region`: su región de AWS.

   En `CognitoUserPool`, redefina las siguientes variables:

   - `PoolId`: su ID de grupo de usuarios de Amazon Cognito.
   - `AppClientId`: el ID de cliente de aplicación.
   - `AppClientSecret`: el secreto del cliente de aplicación.
   - `Region`: su región de AWS.

Para configurar la aplicación de demostración del SDK de Android

Cuando defina las variables de configuración, utilice el formato del los valores de marcador de posición proporcionados en los archivos de configuración.

1. Confirme que ha instalado el SDK para Android para dispositivos Bluetooth de FreeRTOS.
2. Cree una política de AWS IoT en su región (si aún no lo ha hecho).

   > **Note**
   >
   > Esta política es diferente de la política de IAM creada para la identidad autenticada de Amazon Cognito.

   a. Abra la consola de AWS IoT.
   b. En el panel de navegación, elija Secure (Seguridad), elija Policies (Políticas) y, a continuación, elija Create (Crear). Especifique un nombre que identifique la política. En la sección Añadir declaraciones, elija Modo avanzado. Copie y pegue la siguiente política JSON en la ventana del editor de políticas. Sustituya *aws-region* y *aws-account* por su región e ID de cuenta de AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource":"arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Subscribe",
             "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
```

```
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:region:account-id:*"
        }
    ]
}
```

c. Seleccione Crear.

3. Abra https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants.java y vuelva a definir las siguientes variables:

   - `AWS_IOT_POLICY_NAME`: el nombre de la política de AWS IoT.
   - `AWS_IOT_REGION`: su región de AWS.

4. Abra https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/res/raw/awsconfiguration.json.

   En `CognitoIdentity`, redefina las siguientes variables:

   - `PoolId`: su ID de grupo de identidades de Amazon Cognito.
   - `Region`: su región de AWS.

   En `CognitoUserPool`, redefina las siguientes variables:

   - `PoolId`: su ID de grupo de usuarios de Amazon Cognito.
   - `AppClientId`: el ID de cliente de aplicación.
   - `AppClientSecret`: el secreto del cliente de aplicación.
   - `Region`: su región de AWS.

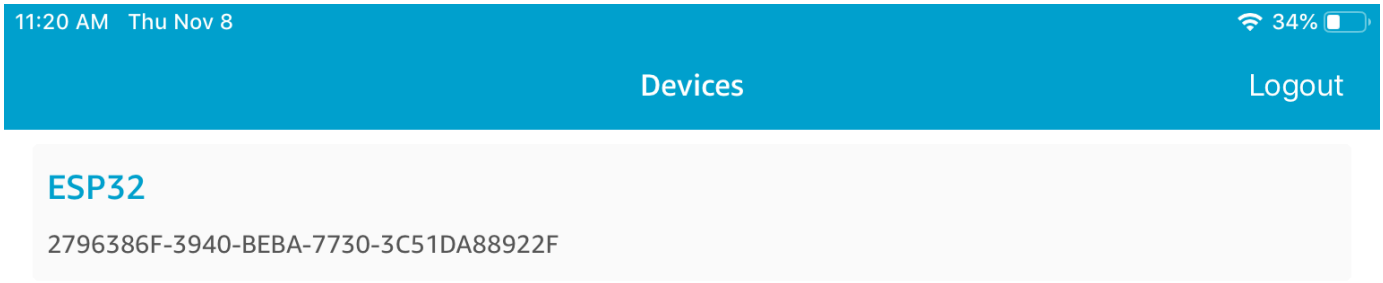
Para detectar y establecer conexiones seguras con su microcontrolador a través de Bluetooth de bajo consumo

1. Para emparejar el microcontrolador y el dispositivo móvil de forma segura (paso 6), necesita un emulador de terminal serie con funciones de entrada y salida (como TeraTerm). Configure el terminal para que se conecte a la placa mediante una conexión en serie, tal como se indica en Instalación de un emulador de terminal.
2. Ejecute el proyecto de demostración de Bluetooth de bajo consumo en su microcontrolador.
3. Ejecute la aplicación de demostración del SDK para móviles de Bluetooth de bajo consumo en su dispositivo móvil.

Para iniciar la aplicación de demostración en el SDK de Android desde la línea de comandos, ejecute el siguiente comando:

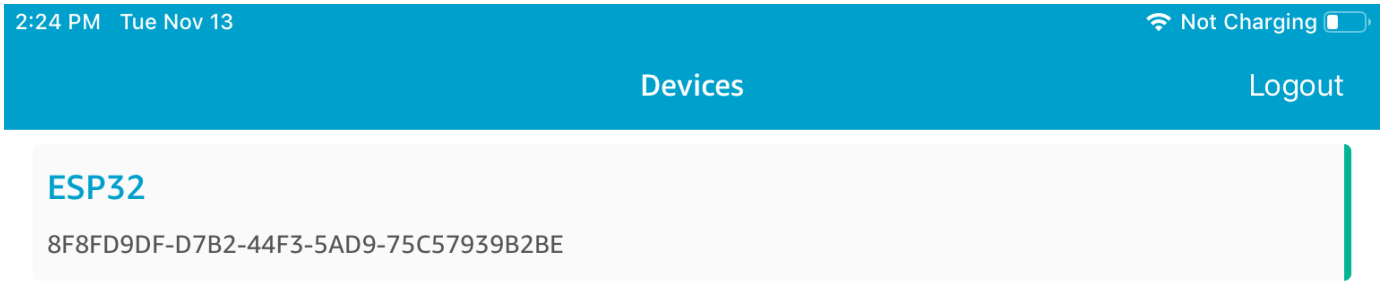
```
$ ./gradlew installDebug
```

4. Confirme que su microcontrolador aparece en Devices (Dispositivos) en la aplicación de demostración del SDK para móviles de Bluetooth de bajo consumo.

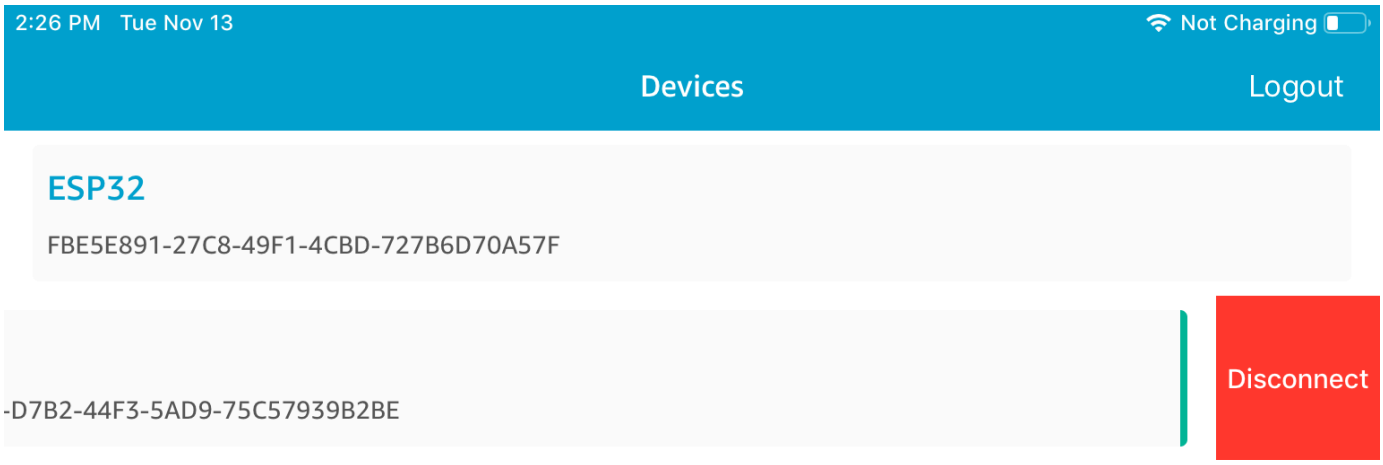


> **Note**
>
> Todos los dispositivos con FreeRTOS y el servicio de información del dispositivo (*freertos*/.../device_information) que estén dentro del rango aparecen en la lista.

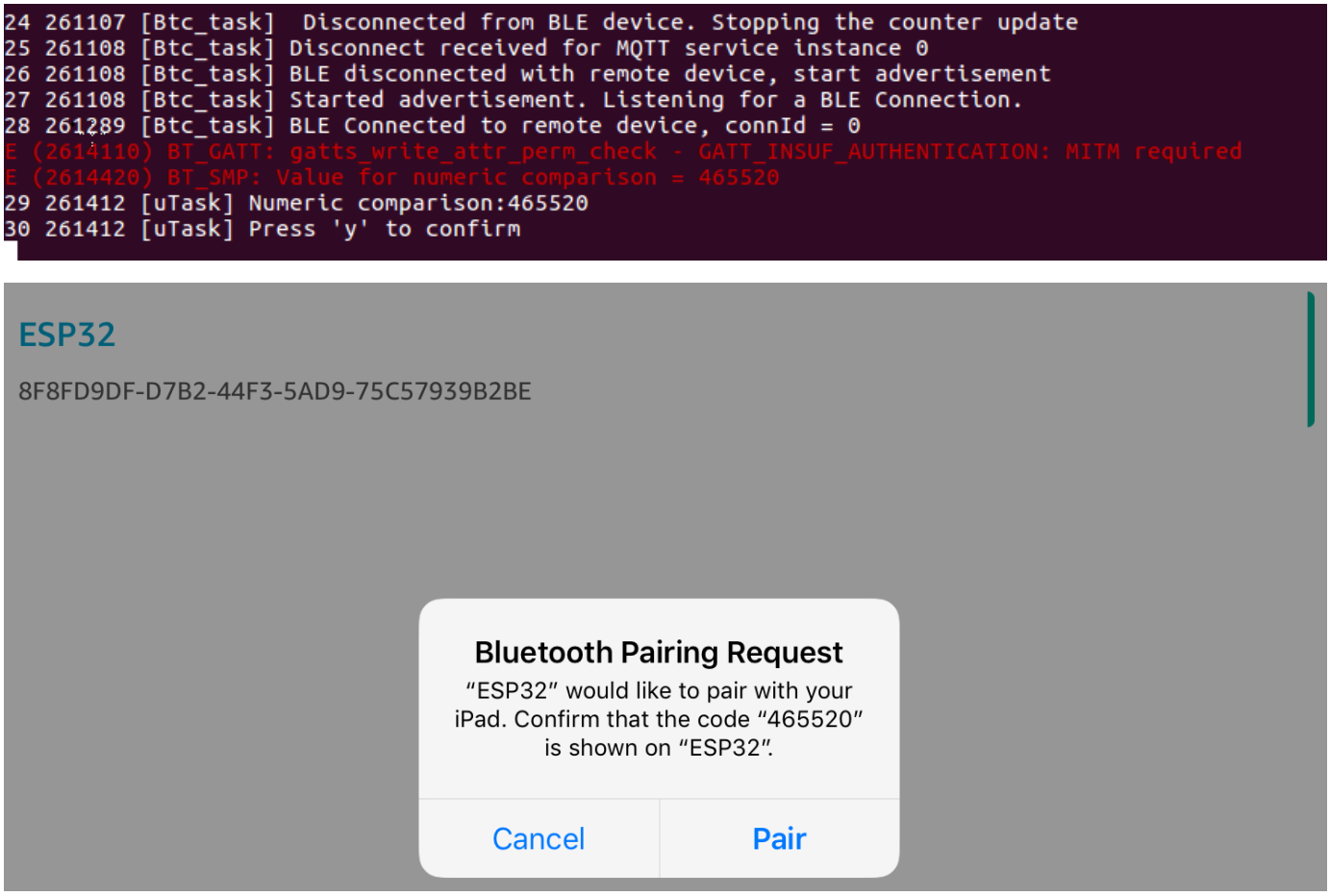
5. Elija su microcontrolador en la lista de dispositivos. La aplicación establece una conexión con la placa y una línea verde aparece junto al dispositivo conectado.



Puede desconectar de su microcontrolador arrastrando la línea a la izquierda.



6. Si se le solicita, empareje el microcontrolador y el dispositivo móvil.



Si el código de comparación numérica es el mismo en ambos dispositivos, empareje los dispositivos.

> **Note**
>
> La aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo utiliza Amazon Cognito para la autenticación del usuario. Asegúrese de que ha configurado un usuario y grupos de identidades de Amazon Cognito y que ha asociado políticas de IAM a identidades autenticadas.

MQTT a través de Bluetooth de bajo consumo

En la demostración de MQTT a través de Bluetooth de bajo consumo, el microcontrolador publica mensajes en la nube de AWS a través de un proxy de MQTT.

Para suscribirse a un tema MQTT de demostración

1. Inicie sesión en la consola de AWS IoT.
2. En el panel de navegación, seleccione Probar y, a continuación, seleccione el cliente de prueba MQTT para abrir el cliente MQTT.
3. En Tema de suscripción, escriba ***thing-name*/example/topic1** y, a continuación, elija Suscribirse al tema.

Si utiliza Bluetooth de bajo consumo para emparejar el microcontrolador con su dispositivo móvil, los mensajes de MQTT se dirigen a través de la aplicación de demostración del SDK para móviles de Bluetooth de bajo consumo en su dispositivo móvil.

Para habilitar la demostración a través de Bluetooth de bajo consumo

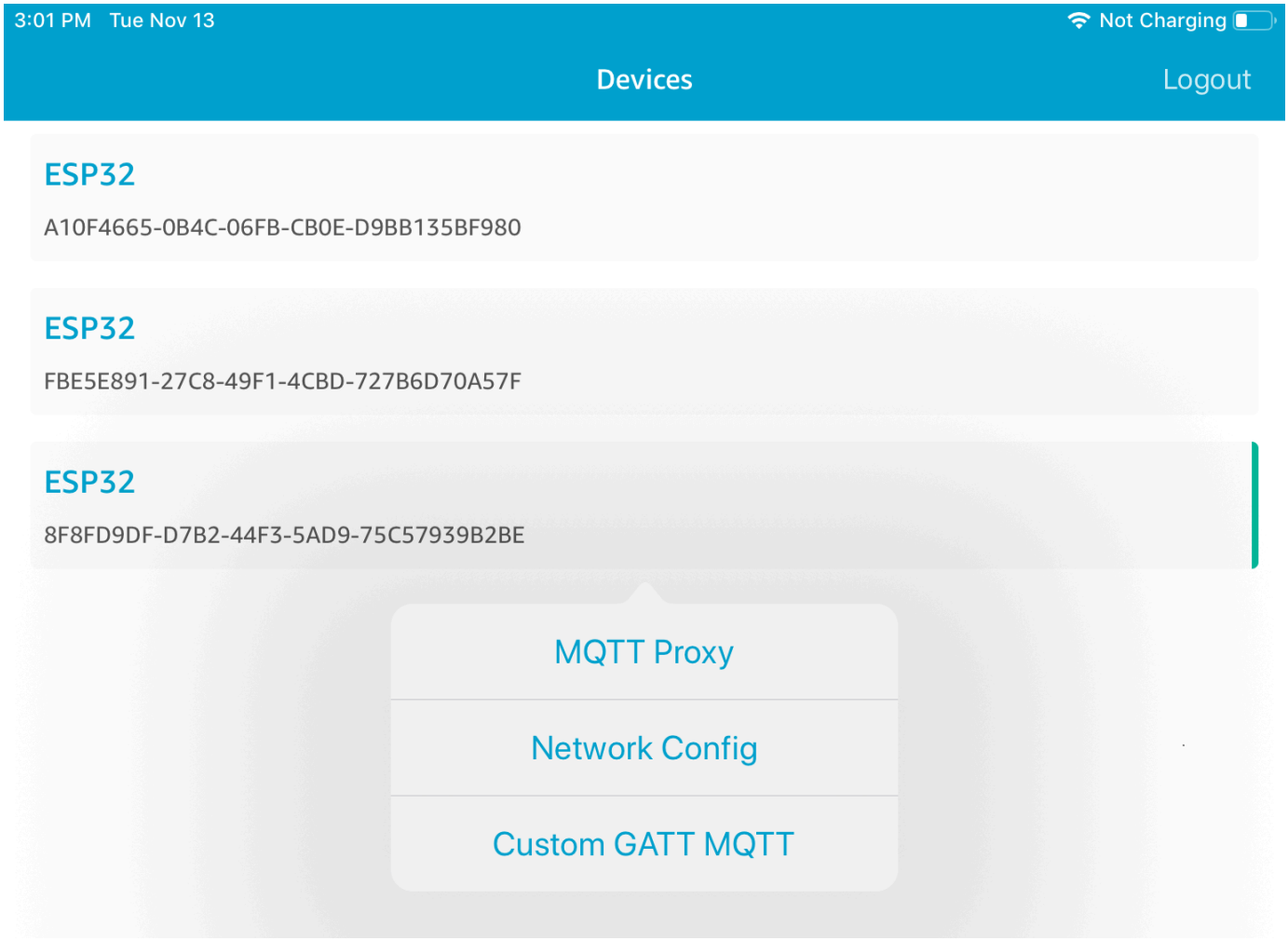
1. Abra `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h` y defina `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`.
2. Abra `demos/include/aws_clientcredential.h` y configure `clientcredentialMQTT_BROKER_ENDPOINT` con el punto de conexión de agente de AWS IoT. Configure `clientcredentialIOT_THING_NAME` con el nombre del objeto del dispositivo microcontrolador BLE. El punto de conexión de agente de AWS IoT se puede obtener desde

la consola de AWS IoT seleccionando Configuración en el panel de navegación izquierdo o mediante la CLI, ejecutando el comando: `aws iot describe-endpoint --endpoint-type=iot:Data-ATS`.

> **Note**
>
> Tanto el punto de conexión de agente de AWS IoT como el nombre del objeto deben estar en la misma región en la que están configurados la identidad y el grupo de usuarios de Cognito.

Para ejecutar la demostración

1. Compile y ejecute el proyecto de demostración en su microcontrolador.
2. Asegúrese de que ha emparejado la placa y el dispositivo móvil mediante Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS.
3. En la lista Devices (Dispositivos) en la aplicación de demostración para móviles, elija el microcontrolador y, a continuación, elija MQTT Proxy (Proxy de MQTT) para abrir la configuración del proxy de MQTT.



4. Después de habilitar el proxy MQTT, los mensajes de MQTT aparecen en el tema *`thing-name`*`/example/topic1` y los datos se imprimen en el terminal UART.

Aprovisionamiento Wi-Fi

El aprovisionamiento Wi-Fi es un servicio de Bluetooth de bajo consumo de FreeRTOS que le permite enviar de forma segura las credenciales de red Wi-Fi desde un dispositivo móvil a un microcontrolador a través de Bluetooth de bajo consumo. El código fuente del servicio de aprovisionamiento wifi se encuentra en *`freertos`*`/.../wifi_provisioning`.

> **Note**
>
> La demostración de aprovisionamiento de Wi-Fi se admite actualmente en Espressif ESP32-DevKitC.

Para habilitar la demostración

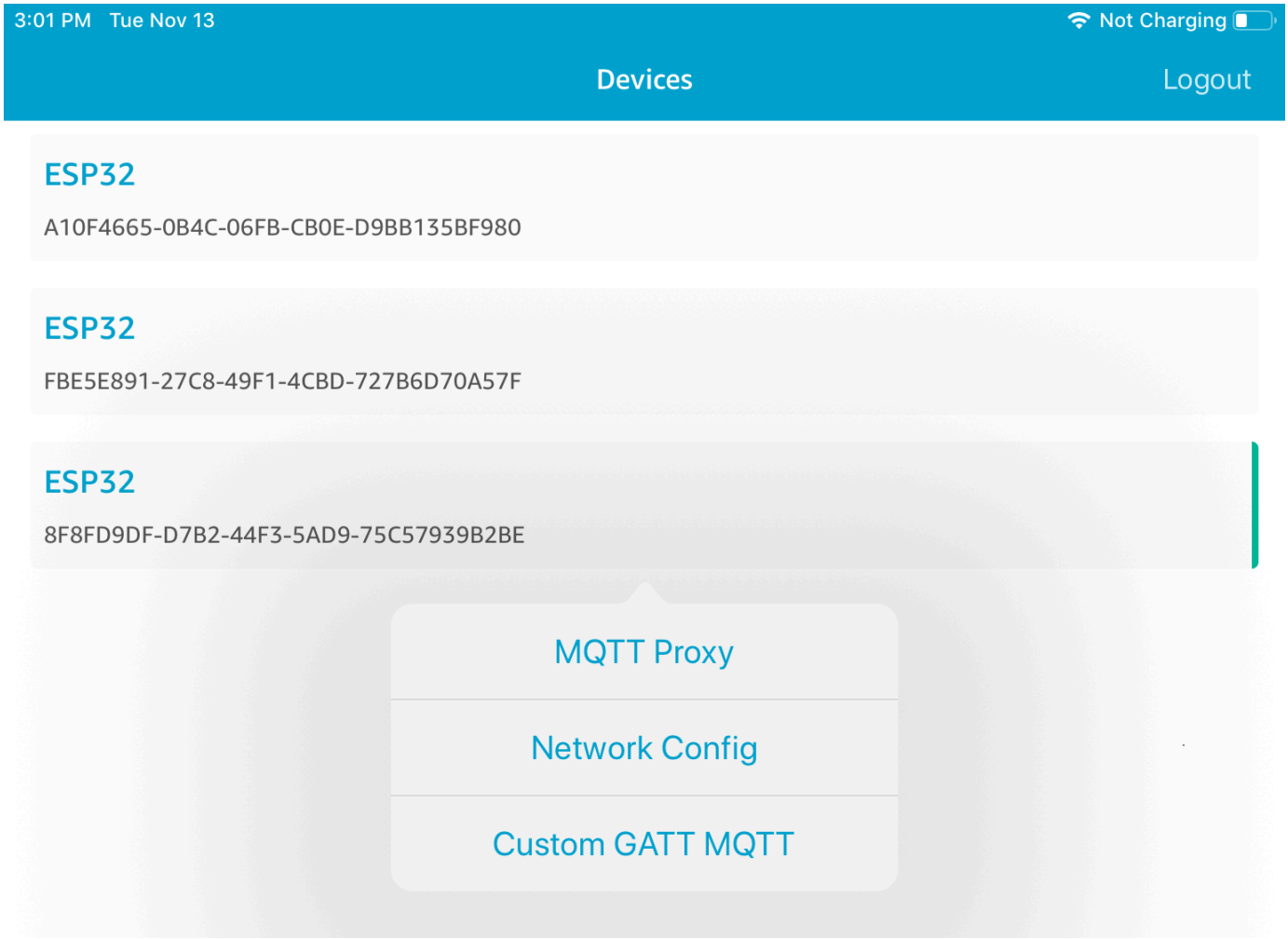
1. Habilite el servicio de aprovisionamiento de Wi-Fi. Abra `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` y establezca `#define IOT_BLE_ENABLE_WIFI_PROVISIONING` en `1` (donde *`vendor`* es el nombre del proveedor y *`board`* es el nombre de la placa que utiliza para ejecutar las demostraciones).

   > **Note**
   >
   > El servicio de aprovisionamiento de Wi-Fi está deshabilitado de forma predeterminada.

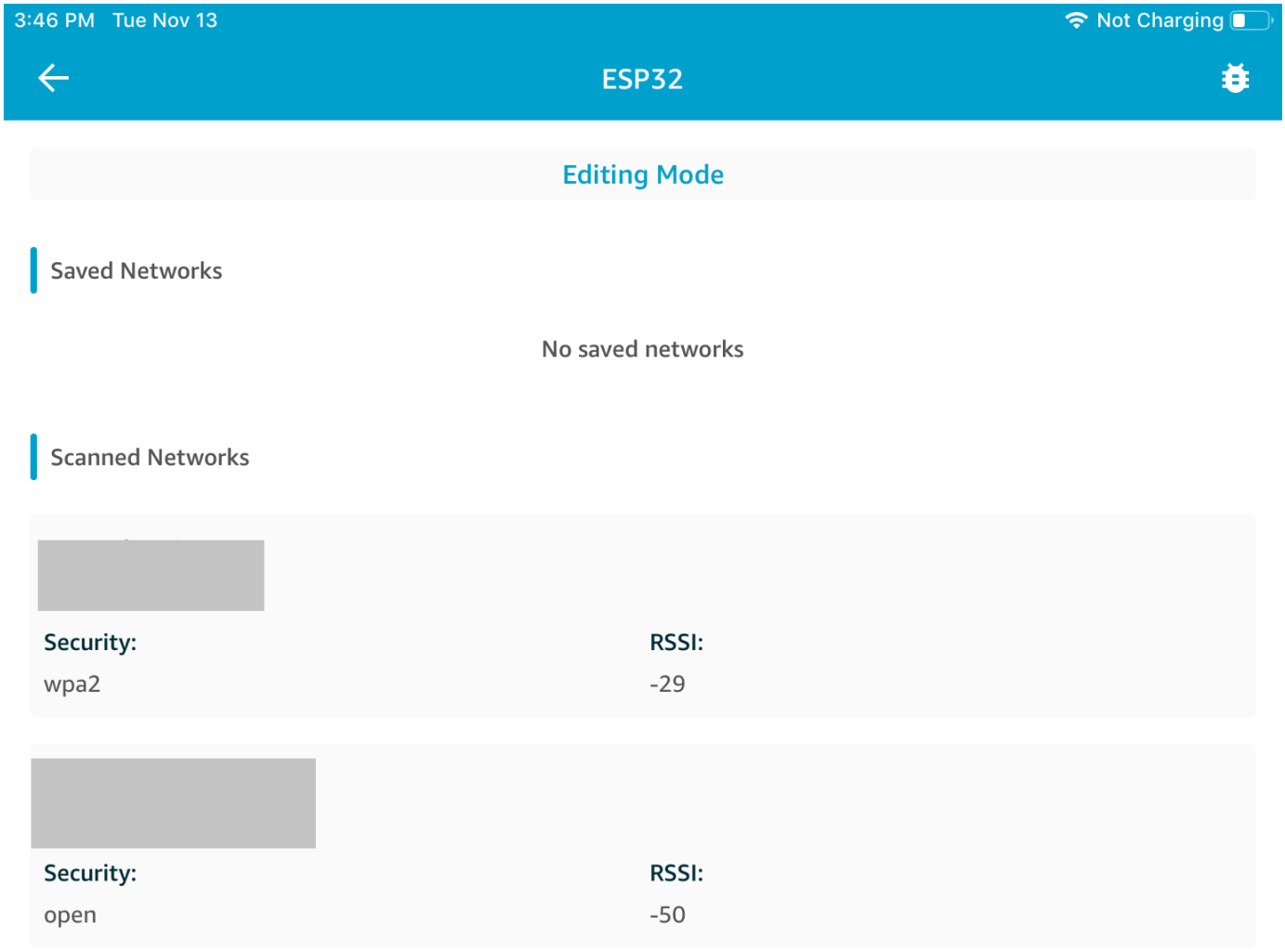
2. Configure el Administrador de red para habilitar Bluetooth de bajo consumo y Wi-Fi.

Para ejecutar la demostración

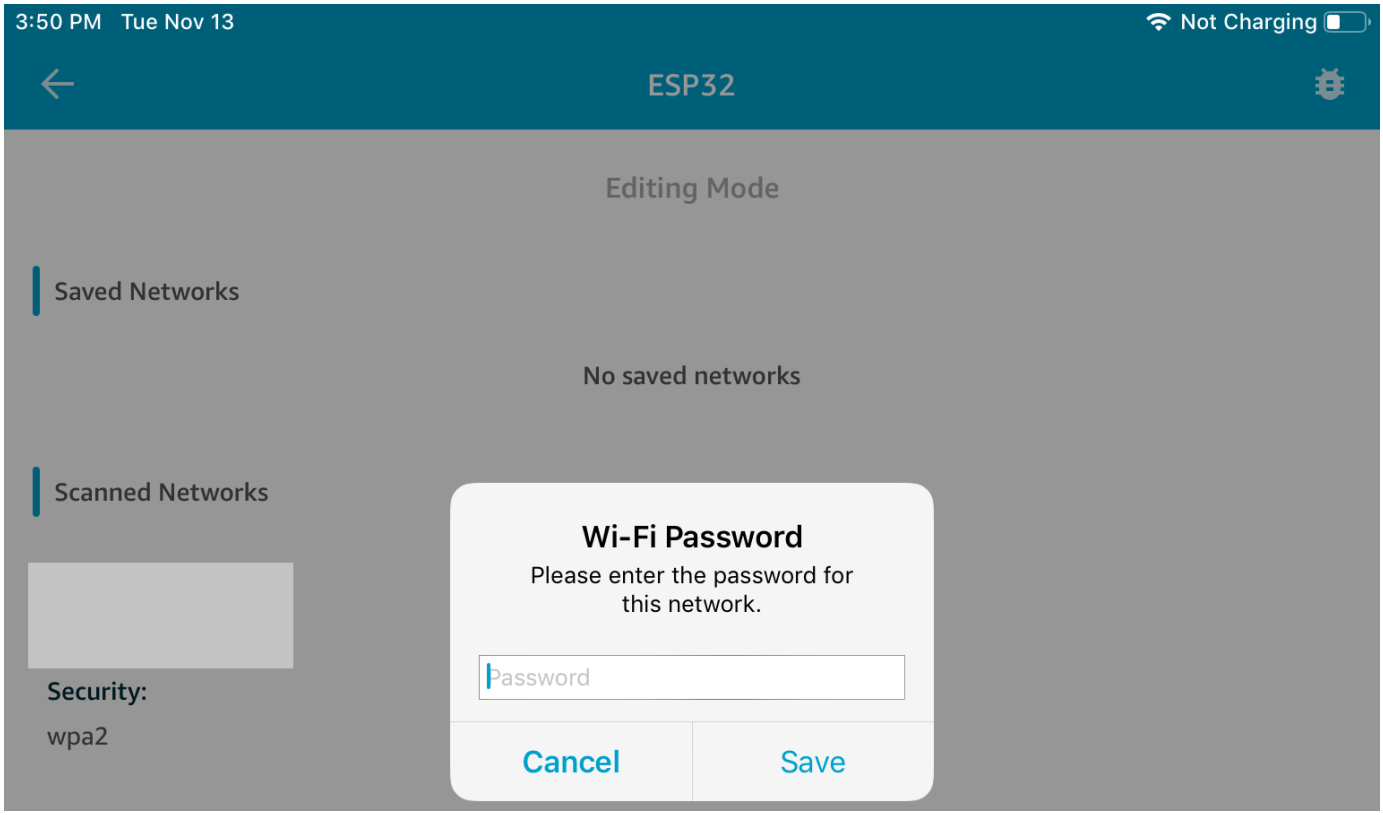
1. Compile y ejecute el proyecto de demostración en su microcontrolador.
2. Asegúrese de que ha emparejado el microcontrolador y el dispositivo móvil mediante Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS.
3. En la lista Devices (Dispositivos) de la aplicación móvil de demostración, elija su microcontrolador y, a continuación, elija Network Config (Configurar red) para abrir los ajustes de configuración de red.



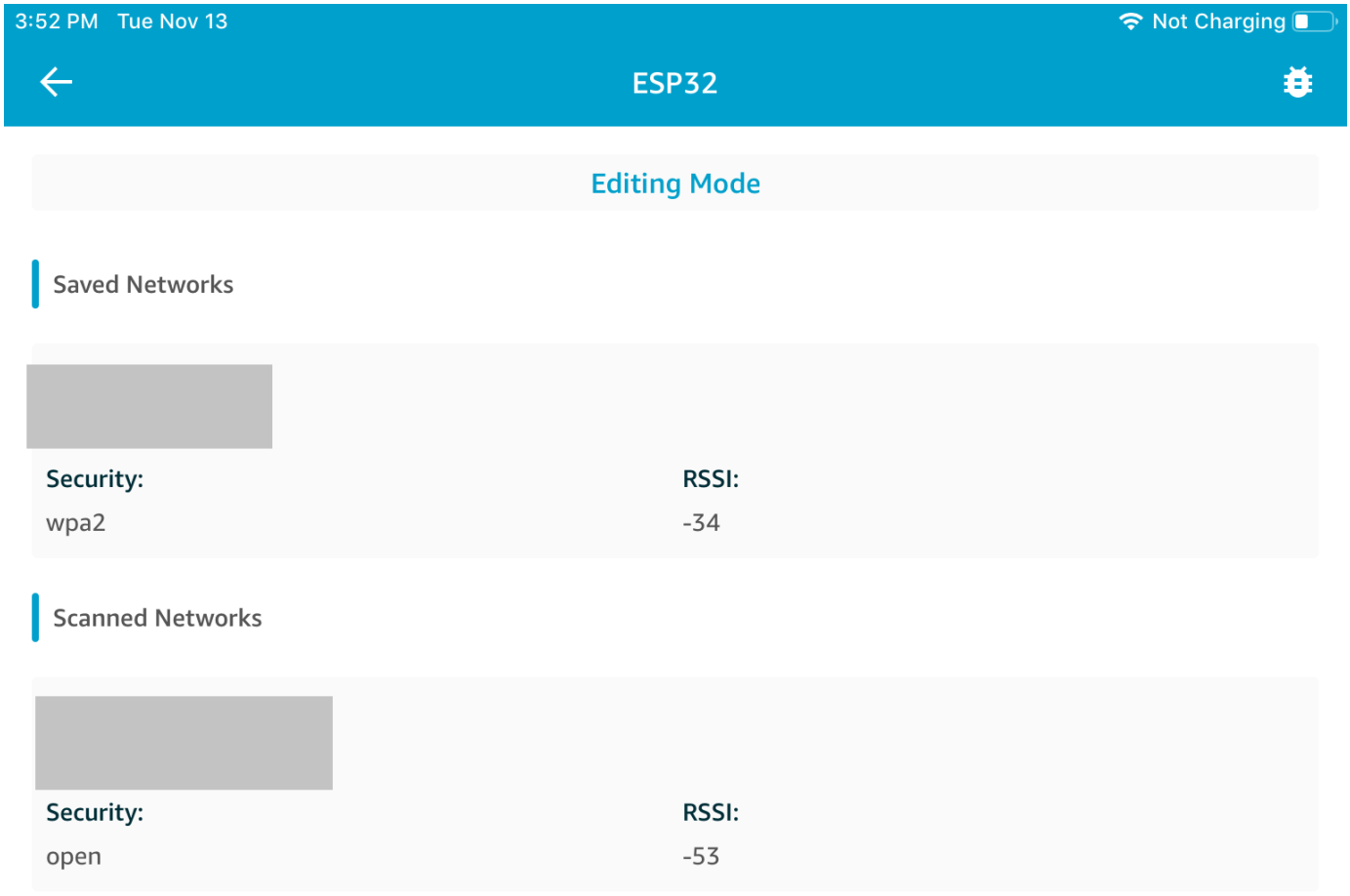
4. Después de elegir Network Config (Config. de red) para la placa, el microcontrolador envía una lista de las redes del entorno al dispositivo móvil. Las redes Wi-Fi disponibles aparecen en una lista en Scanned Networks (Redes analizadas).



En la lista Scanned Networks (Redes analizadas), elija la red y, a continuación, introduzca el SSID y la contraseña, si es necesario.

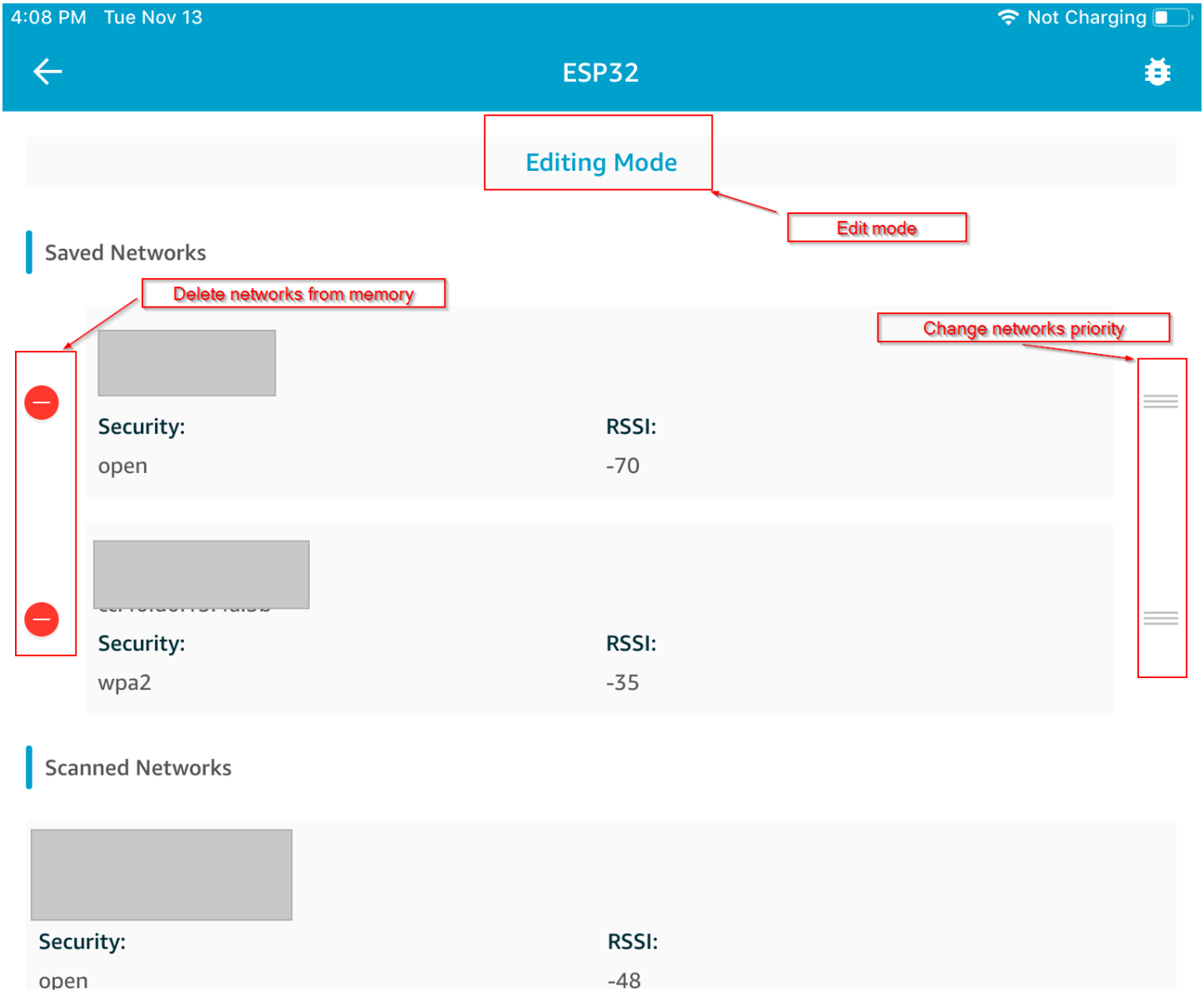


El microcontrolador se conecta a la red y la guarda. La red aparece en Saved Networks (Redes guardadas).



Puede guardar varias redes en la aplicación de demostración para móviles. Al reiniciar la aplicación y la demostración, el microcontrolador se conecta a la primera red guardada, empezando por el principio de la lista Saved Networks (Redes guardadas).

Para cambiar el orden de prioridad de red o eliminar redes, en la página Network Configuration (Configuración de red), seleccione Editing Mode (Modo de edición). Para cambiar el orden de prioridad de red, elija el lado derecho de la red a la que desea dar prioridad y arrastre la red hacia arriba o hacia abajo. Para eliminar una red, elija el botón rojo en el lado izquierdo de la red que desea eliminar.



Servidor de atributos genéricos

En este ejemplo, la aplicación del servidor de atributos genéricos (GATT) de su microcontrolador envía un valor de contador simple a Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS.

Con el SDK para móviles de Bluetooth de bajo consumo, puede crear su propio cliente GATT para un dispositivo móvil que se conecta al servidor de GATT en su microcontrolador y se ejecuta en paralelo con la aplicación de demostración para móviles.

Para habilitar la demostración

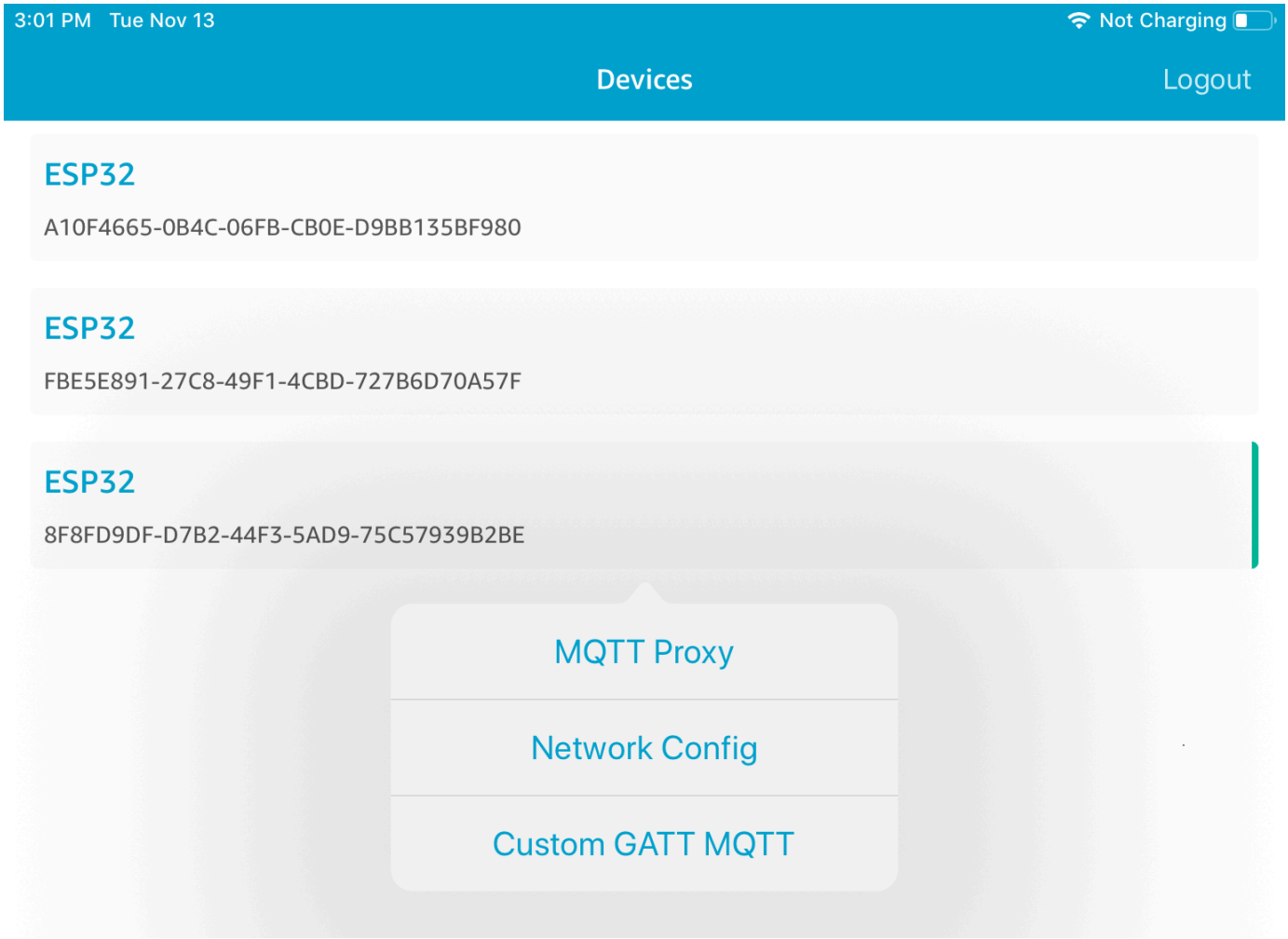
1. Habilite la demostración de GATT de Bluetooth de bajo consumo. En `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` (donde *vendor* es el nombre del proveedor y *board* es el nombre de la placa que utiliza para ejecutar las demostraciones), agregue `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` a la lista de instrucciones define.

   > **Note**
   >
   > La demostración de GATT de Bluetooth de bajo consumo está deshabilitada de forma predeterminada.

2. Abra *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` y defina `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`.

Para ejecutar la demostración

1. Compile y ejecute el proyecto de demostración en su microcontrolador.
2. Asegúrese de que ha emparejado la placa y el dispositivo móvil mediante Aplicación de demostración de SDK para móviles de Bluetooth de bajo consumo de FreeRTOS.
3. En la lista Devices (Dispositivos) de la aplicación, elija la placa y, a continuación, elija MQTT Proxy (Proxy de MQTT) para abrir la las opciones del proxy de MQTT.



4. Vuelva a la lista Devices (Dispositivos), elija la placa y, a continuación, seleccione Custom GATT MQTT (MQTT de GATT personalizado) para abrir las opciones de servicio GATT personalizado.
5. Elija Start Counter (Iniciar contador) para empezar a publicar datos en el tema de MQTT ***your-thing-name*/example/topic**.

   Después de habilitar el proxy de MQTT, Hello World e incrementar el contador, los mensajes aparecen en el tema *your-thing-name*/example/topic.

Cargador de arranque de demostración para la placa de desarrollo Curiosity PIC32MZEF de Microchip

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto

FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Note

De acuerdo con Microchip, eliminaremos Curiosity PIC32MZEF (DM320104) de la rama principal del repositorio de integración de referencias de FreeRTOS y ya no lo incluiremos en las nuevas versiones. Microchip ha publicado un aviso oficial en el que indica que PIC32MZEF (DM320104) ya no se recomienda para los nuevos diseños. Aún se puede acceder a los proyectos y al código fuente de PIC32MZEF a través de las etiquetas de las versiones anteriores. Microchip recomienda a los clientes que utilicen la placa de desarrollo PIC32MZ-EF-2.0 (DM320209) de Curiosity para los nuevos diseños. La plataforma PIC32MZv1 todavía se encuentra en la versión 202012.00 del repositorio de integración de referencias de FreeRTOS. Sin embargo, la plataforma ya no es compatible con la versión 202107.00 de la referencia de FreeRTOS.

Este cargador de arranque de demostración implementa la comprobación de la versión de firmware, la verificación de firma criptográfica y el autodiagnóstico de la aplicación. Estas capacidades admiten actualizaciones de firmware inalámbricas (OTA) para FreeRTOS.

La verificación de firmware incluye la verificación de la autenticidad y la integridad del firmware nuevo recibido de forma inalámbrica. El cargador de arranque verifica la firma criptográfica de la aplicación antes del arranque. La demostración utiliza el Algoritmo de firma digital de curva elíptica (ECDSA) a través de SHA-256. Las utilidades proporcionadas se pueden utilizar para generar una aplicación firmada que se pueden actualizar en el dispositivo.

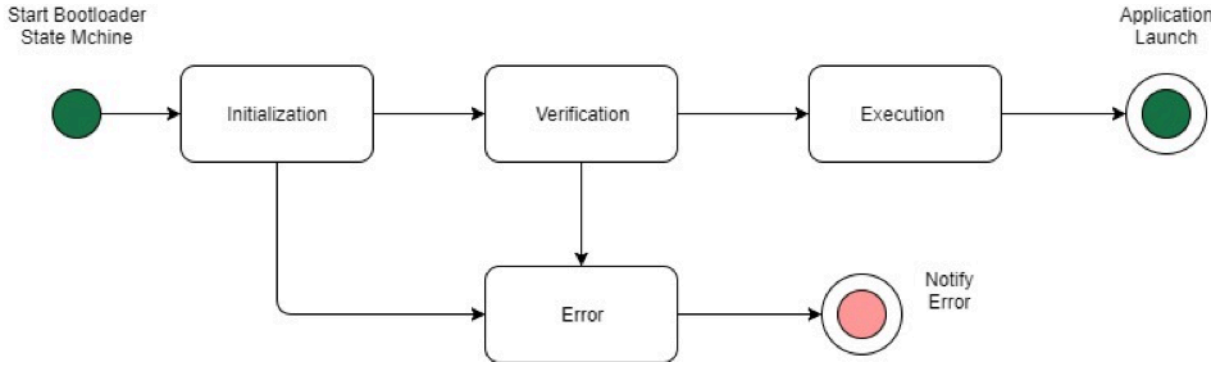
El cargador de arranque es compatible con las siguientes características necesarias para OTA:

- Mantiene imágenes de la aplicación en el dispositivo y cambia entre ellas.
- Permite la ejecución del autodiagnóstico de una imagen recibida a través de OTA y la reversión en caso de error.
- Comprueba la firma y la versión de la imagen de actualización OTA.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Estados del cargador de arranque

El proceso del cargador de arranque se muestra en la siguiente máquina de estado.



En la tabla siguiente se describen los estados del cargador de arranque.

Estado del cargador de arranque	Descripción
Inicialización	El cargador de arranque se encuentra en el estado de inicialización.
Verification (Verificación)	El cargador de arranque está verificando las imágenes presentes en el dispositivo.
Execute Image (Ejecutar imagen)	El cargador de arranque está lanzando la imagen seleccionada.
Execute Default (Ejecutar predeterminada)	El cargador de arranque está lanzando la imagen predeterminada.
Error	El cargador de arranque se encuentra en el estado de error.

En el diagrama anterior, se muestran `Execute Image` y `Execute Default` como el estado `Execution`.

Bootloader Execution State (Estado de ejecución del cargador de arranque)

El cargador de arranque se encuentra en el estado `Execution` y está listo para lanzar la imagen verificada seleccionada. Si la imagen que se van a lanzar se encuentra en el banco superior, los bancos se intercambian antes de ejecutar la imagen, ya que la aplicación siempre se crea para el banco inferior.

Bootloader Default Execution State (Estado de ejecución predeterminado del cargador de arranque)

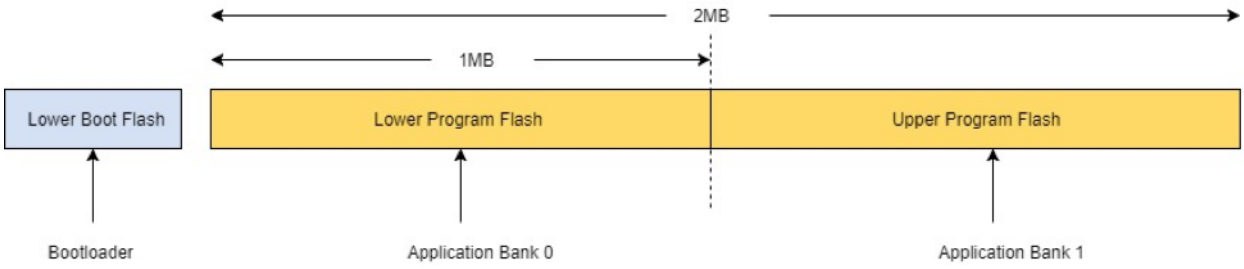
Si la opción de configuración para lanzar la imagen predeterminada está habilitada, el cargador de arranque lanza la aplicación desde una dirección de ejecución predeterminada. Esta opción debe estar deshabilitada, excepto durante la depuración.

Bootloader Error State (Estado de error del cargador de arranque)

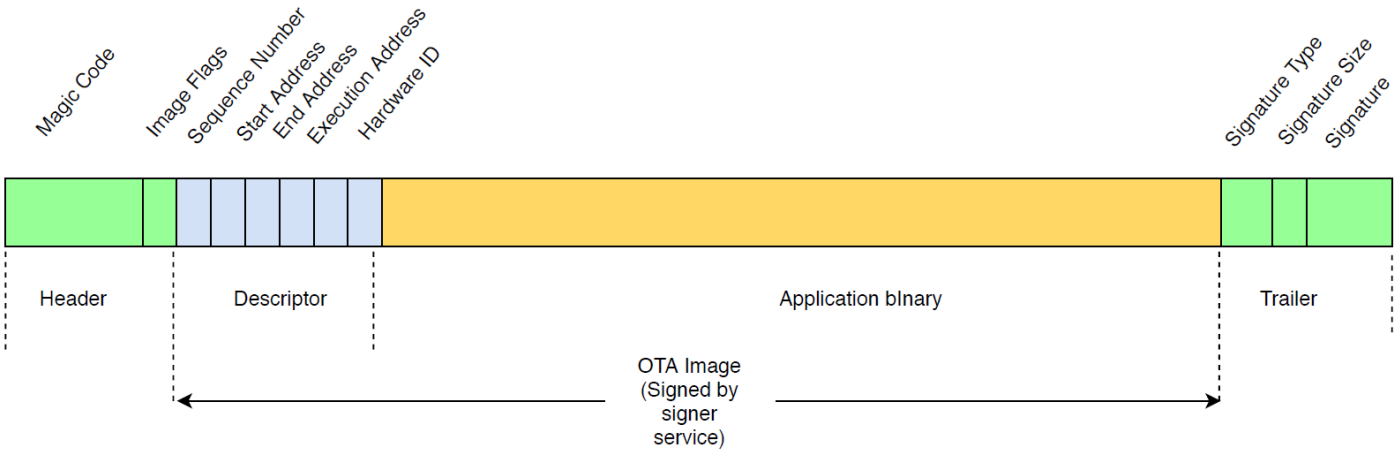
El cargador de arranque se encuentra en un estado de error y no hay imágenes válidas presentes en el dispositivo. El cargador de arranque debe notificar al usuario. La implementación predeterminada envía un mensaje de registro a la consola y el LED parpadea en el tablero de forma indefinida.

Dispositivo flash

La plataforma de la Curiosity PIC32MZEF de Microchip contiene un flash de programa interno de dos megabytes (MB) dividido en dos bancos. Admite el intercambio de mapas de memoria entre estos dos bancos y actualizaciones directas. El cargador de arranque de demostración se programa en una región flash del cargador inferior separada.



Estructura de la imagen de la aplicación



El diagrama muestra los componentes principales de la imagen de aplicación almacenados en cada banco del dispositivo.

Componente	Tamaño (en bytes)
Encabezado de la imagen	8 bytes
Descriptor de la imagen	24 bytes
Binario de la aplicación	< 1 MB - (324)
Trailer	292 bytes

Encabezado de la imagen

Las imágenes de la aplicación en el dispositivo deben comenzar con un encabezado que consta de un código mágico y marcas de imágenes.

Campos del encabezado	Tamaño (en bytes)
Código mágico	7 bytes
Marcadores de imágenes	1 byte

Código mágico

La imagen en el dispositivo Flash debe empezar con un código mágico. El código mágico predeterminado es `@AFRTOS`. El cargador de arranque comprueba si hay un código mágico válido presente antes de arrancar la imagen. Esta es el primer paso de la verificación.

Marcadores de imágenes

Los marcadores de imágenes se utilizan para almacenar el estado de las imágenes de la aplicación. Los marcadores se utilizan en el proceso de OTA. Los marcadores de imágenes de ambos bancos determinan el estado del dispositivo. Si la imagen de ejecución se marca como pendiente de confirmación, significa que el dispositivo está en la fase de autodiagnóstico OTA. Aunque las imágenes en los dispositivos se marcan como válidas, pasan por los mismos pasos de verificación en cada arranque. Si una imagen se marca como nueva, el cargador de arranque la marca como pendiente de confirmación y la lanza para el autodiagnóstico después de la verificación. El cargador de arranque también inicializa e inicia el temporizador de vigilancia de modo que si se produce un error en el autodiagnóstico de la nueva imagen OTA, el dispositivo se reinicia y el cargador de arranque rechaza la imagen eliminándola y ejecuta la imagen válida anterior.

El dispositivo solo puede tener una imagen válida. La otra imagen puede ser una imagen OTA nueva o una pendiente de confirmación (autodiagnóstico). Después de una actualización OTA correcta, la imagen anterior se elimina del dispositivo.

Estado	Valor	Descripción
New image (Nueva imagen)	0xFF	La imagen de aplicación es nueva y no se ha ejecutado nunca.
Commit pending (Confirmación pendiente)	0xFE	La imagen de la aplicación está marcada para la ejecución de las pruebas.
Valid (Válido)	0xFC	La imagen de la aplicación está marcada como válida y confirmada.
Invalid (No válido)	0xF8	La imagen de la aplicación está marcada como no válido.

Descriptor de la imagen

La imagen de aplicación en el dispositivo flash debe contener el descriptor de la tras el encabezado de la imagen. El descriptor de la imagen es generado por una utilidad posterior a la compilación que utiliza archivos de configuración (`ota-descriptor.config`) para generar el descriptor adecuado y lo añade al binario de la aplicación. El resultado de este paso posterior a la compilación es la imagen binaria que se puede utilizar para OTA.

Campo del descriptor	Tamaño (en bytes)
Sequence Number (Número de secuencia)	4 bytes
Start Address (Dirección de inicio)	4 bytes
End Address (Dirección final)	4 bytes
Execution Address (Dirección de ejecución)	4 bytes
Hardware ID (ID de hardware)	4 bytes
Reserved (Reservado)	4 bytes

Sequence Number (Número de secuencia)

El número de secuencia debe incrementarse antes de crear una nueva imagen OTA. Consulte el archivo `ota-descriptor.config`. El cargador de arranque utiliza este número para determinar la imagen que se va a arrancar. Los valores válidos son de 1 a 4294967295.

Start Address (Dirección de inicio)

La dirección de partida de la imagen de aplicación en el dispositivo. Como el descriptor de la imagen se anexa al código binario de la aplicación, esta dirección es el principio del descriptor de la imagen.

End Address (Dirección final)

La dirección final de la imagen de la aplicación en el dispositivo, sin incluir el tráiler de imágenes.

Execution Address (Dirección de ejecución)

La dirección de ejecución de la imagen.

Hardware ID (ID de hardware)

Un ID de hardware único utilizado por el cargador de arranque para verificar que la imagen de OTA se ha creado para la plataforma correcta.

Reserved (Reservado)

Esto se reserva para un uso ulterior.

Tráiler de imágenes

El tráiler de imágenes se añade al código binario de la aplicación. Contiene la cadena del tipo de firma, el tamaño de la firma y la firma de la imagen.

Campo del tráiler	Tamaño (en bytes)
Signature Type (Tipo de firma)	32 bytes
Signature Size (Tamaño de la firma)	4 bytes
Firma	256 bytes

Signature Type (Tipo de firma)

El tipo de firma es una cadena que representa el algoritmo criptográfico que se está utilizando y sirve como un marcador para el tráiler. El cargador de arranque admite el Algoritmo de firma digital de curva elíptica (ECDSA). El valor predeterminado es sig-sha256-ecdsa.

Signature Size (Tamaño de la firma)

El tamaño en bytes de la firma criptográfica.

Firma

La firma criptográfica del código binario de la aplicación anexado al descriptor de la imagen.

Configuración del cargador de arranque

Las opciones de configuración básica del cargador de arranque se proporcionan en *freertos*/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h. Se proporcionan algunas opciones para fines de depuración solamente.

Enable Default Start (Habilitar inicio predeterminado)

Habilita la ejecución de la aplicación en la dirección predeterminada y debe estar habilitado solo para la depuración. La imagen se ejecuta desde la dirección predeterminada sin ningún tipo de verificación.

Enable Crypto Signature Verification (Habilitar verificación de la firma criptográfica)

Habilita la verificación de firma criptográfica durante el arranque. Las imágenes con error se borran del dispositivo. Esta opción se ofrece solo para fines de depuración y debe permanecer habilitada en la producción.

Erase Invalid Image (Borrar imagen no válida)

Permite borrar un banco completo si se produce un error en la verificación de imagen de dicho banco. La opción se ofrece para fines de depuración y debe permanecer habilitada en la producción.

Enable Hardware ID Verification (Habilitar verificación de ID de hardware)

Habilita la verificación de ID de hardware en el descriptor de la imagen OTA y el ID de hardware programado en el cargador de arranque. Esto es opcional y se puede deshabilitar si no es necesario verificar el ID de hardware.

Enable Address Verification (Habilitar verificación de dirección)

Habilita la verificación de las direcciones de inicio, finalización y ejecución en el descriptor de la imagen OTA. Recomendamos que mantenga esta opción habilitada.

Creación del cargador de arranque

El cargador de arranque de demostración se incluye como un proyecto que se puede cargar en el proyecto `aws_demos` ubicado en *`freertos`*`/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/`, en el repositorio de códigos fuente de FreeRTOS. Cuando se crea el proyecto `aws_demos`, crea el cargador de arranque en primer lugar, seguido de la aplicación. El resultado final es una imagen hexadecimal unificada que incluye el cargador de arranque y la aplicación. La utilidad `factory_image_generator.py` se suministra para generar una imagen hexadecimal unificada con firma criptográfica. Los scripts de utilidades del cargador de arranque se encuentran en *`freertos`*`/demos/ota/bootloader/utility/`.

Paso de compilación previa del cargador de arranque

Este paso de compilación previa ejecuta un script de utilidad llamado `codesigner_cert_utility.py` que extrae la clave pública del certificado de firma de código y genera un archivo de encabezado C que contiene la clave pública en formato codificado de Notación de Sintaxis Abstracta Uno (ASN.1). Este encabezado se compila en el proyecto del cargador de arranque. El encabezado generado contiene dos constantes: una matriz de la clave pública y la longitud de la clave. El proyecto del cargador de arranque también se puede crear sin `aws_demos` y se puede depurar como una aplicación normal.

Demostración de AWS IoT Device Defender

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

Esta demostración le muestra cómo utilizar la biblioteca de AWS IoT Device Defender para conectarse a AWS IoT Device Defender. La demostración utiliza la biblioteca coreMQTT para establecer una conexión MQTT con TLS (autenticación mutua) con el agente de MQTT de AWS IoT y la biblioteca coreJSON para validar y analizar las respuestas recibidas del servicio AWS IoT Device Defender. La demostración muestra cómo crear un informe con formato JSON utilizando las métricas recopiladas del dispositivo y cómo enviar el informe creado al servicio AWS IoT Device Defender. La demostración también muestra cómo registrar una función de devolución de llamada en la biblioteca coreMQTT para gestionar las respuestas del servicio AWS IoT Device Defender para confirmar si un informe enviado se ha aceptado o rechazado.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Funcionalidad

Esta demostración crea una tarea de aplicación única que demuestra cómo recopilar métricas, crear un informe de Device Defender en formato JSON y enviarlo al servicio AWS IoT Device Defender a través de una conexión MQTT segura al agente de MQTT de AWS IoT. La demostración incluye las métricas de red estándar, así como métricas personalizadas. Para las métricas personalizadas, la demostración incluye:

- Una métrica denominada "`task_numbers`", que es una lista de identificadores de tareas de FreeRTOS. El tipo de esta métrica es "lista de números".
- Una métrica denominada "`stack_high_water_mark`", que es el límite máximo de pila para la tarea de la aplicación de demostración. El tipo de esta métrica es "número".

La forma en que recopilamos las métricas de red depende de la pila de TCP/IP que se utilice. Para FreeRTOS+TCP y las configuraciones lwIP compatibles, ofrecemos implementaciones de recopilación de métricas que recopilan métricas reales del dispositivo y las envían al informe AWS IoT Device Defender. Puede encontrar las implementaciones para FreeRTOS+TCP e lwIP en GitHub.

Para las placas que utilizan cualquier otra pila de TCP/IP, se proporcionan definiciones simuladas de las funciones de recopilación de métricas que devuelven ceros para todas las métricas de red. Implemente las funciones de *freertos*`/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c` para su pila de red para enviar métricas reales. El archivo también está disponible en el sitio web de GitHub.

Para el ESP32, la configuración de lwIP predeterminada no utiliza el bloqueo de núcleos y, por lo tanto, la demostración utilizará métricas simuladas. Si desea utilizar la implementación de recopilación de métricas de lwIP de referencia, defina las siguientes macros en `lwiopts.h`:

```
#define LINK_SPEED_OF_YOUR_NETIF_IN_BPS 0
#define LWIP_TCPIP_CORE_LOCKING         1
#define LWIP_STATS                      1
#define MIB2_STATS                      1
```

A continuación se muestra una salida de ejemplo al ejecutar la demostración.

```
24 1682 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1682 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1682 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1682 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1722 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1722 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1722 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2322 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2322 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2382 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2382 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2382 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 2982 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1109,"v": "1.0"},"met": {"tp": {"pts": [{"pt": 33251}],"t": 1},
"up": {"pts": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [{"lp": 33251,"rad": "44.236.152.27:888337 2
982 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


38 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
39 3102 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
40 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
41 3102 [iot_thread] [INFO] PUBACK received for packet id 3.


42 3102 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


43 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
44 3102 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
45 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
46 3502 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


47 3542 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
48 3542 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


49 4142 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


50 4202 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
51 4202 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


52 4802 [iot_thread] [INFO] Disconnected from the broker.
53 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::before::bytes::2088152

54 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::after::bytes::1985556

55 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::before::bytes::1908

56 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::after::bytes::1908

57 5802 [iot_thread] [INFO ][DEMO][5802] Demo completed successfully.

58 5804 [iot_thread] [INFO ][INIT][5804] SDK cleanup done.

59 5804 [iot_thread] [INFO ][DEMO][5804] -------DEMO FINISHED-------
```

Si la placa no utiliza FreeRTOS+TCP o una configuración lwIP compatible, la salida tendrá el siguiente aspecto.

```
24 1716 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1716 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1716 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1716
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1756 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1756 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1756 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2356 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2356
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2436 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2436 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2436 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 3036 [iot_thread] [ERROR] Using stub definition of GetNetworkStats! Please implement for your network stack to get correct m
etrics.
37 3036 [iot_thread] [ERROR] Using stub definition of GetOpenTcpPorts! Please implement for your network stack to get correct m
etrics.
38 3036 [iot_thread] [ERROR] Using stub definition of GetOpenUdpPorts! Please implement for your network stack to get correct m
etrics.
39 3036 [iot_thread] [ERROR] Using stub definition of GetEstablishedConnections! Please implement for your network stack to get
 correct metrics.
40 3036 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1079,"v": "1.0"},"met": {"tp": {"pts": [],"t": 0},"up": {"pts
": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [],"t": 0}}},"cmet": {"stack_high_water_mark": [{"num
41 3036 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


42 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
43 3196 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
44 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
45 3196 [iot_thread] [INFO] PUBACK received for packet id 3.


46 3196 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


47 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
48 3196 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
49 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
50 3596 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


51 3656 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
52 3656 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


53 4256 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


54 4336 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
55 4336 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


56 4936 [iot_thread] [INFO] Disconnected from the broker.
57 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::before::bytes::2088152

58 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::after::bytes::1985556

59 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::before::bytes::1908

60 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::after::bytes::1908

61 5936 [iot_thread] [INFO ][DEMO][5936] Demo completed successfully.

62 5938 [iot_thread] [INFO ][INIT][5938] SDK cleanup done.

63 5938 [iot_thread] [INFO ][DEMO][5938] -------DEMO FINISHED-------
```

El código fuente de la demostración está en su directorio *freertos*/demos/device_defender_for_aws/ de descargas o en el sitio web de GitHub.

Suscripción a temas de AWS IoT Device Defender

La función subscribeToDefenderTopics se suscribe a los temas de MQTT sobre los que se recibirán las respuestas a los informes publicados de Device Defender. Utiliza la macro `DEFENDER_API_JSON_ACCEPTED` para crear la cadena de temas en la que se reciben las respuestas a los informes aceptados por Device Defender. Utiliza la macro `DEFENDER_API_JSON_REJECTED` para crear la cadena de temas en la que se reciben las respuestas a los informes rechazados por Device Defender.

Recopilación de métricas de dispositivos

La función collectDeviceMetrics recopila métricas de red mediante las funciones definidas en `metrics_collector.h`. Las métricas recopiladas son la cantidad de bytes y paquetes enviados y recibidos, los puertos TCP abiertos, los puertos UDP abiertos y las conexiones TCP establecidas.

Generación del informe de AWS IoT Device Defender

La función generateDeviceMetricsReport genera un informe de Device Defender mediante la función definida en `report_builder.h`. Esta función toma las métricas de red y un búfer, crea un documento JSON en el formato esperado en AWS IoT Device Defender y lo escribe en el búfer proporcionado. El formato del documento JSON esperado en AWS IoT Device Defender se especifica en Métricas del lado del dispositivo en la Guía para desarrolladores de AWS IoT.

Publicación del informe de AWS IoT Device Defender

El informe de AWS IoT Device Defender se publica sobre el tema MQTT para la publicación de informes de AWS IoT Device Defender de JSON. El informe se crea con la macro `DEFENDER_API_JSON_PUBLISH`, como se muestra en este fragmento de código del sitio web de GitHub.

Devolución de llamada para gestionar las respuestas

La función publishCallback gestiona los mensajes de publicación de MQTT entrantes. Utiliza la API `Defender_MatchTopic` de la biblioteca de AWS IoT Device Defender para comprobar si el mensaje de MQTT entrante proviene del servicio de AWS IoT Device Defender. Si el mensaje proviene del servicio AWS IoT Device Defender, analiza la respuesta JSON recibida y extrae el ID del informe en la respuesta. A continuación, se comprueba que el ID del informe es el mismo que el enviado en el informe.

Aplicación de demostración de detección de AWS IoT Greengrass V1

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Antes de ejecutar la demostración de detección de AWS IoT Greengrass para FreeRTOS, debe configurar AWS, AWS IoT Greengrass y AWS IoT. Para configurar AWS, siga las instrucciones de Configuración de su cuenta de AWS y los permisos. Para configurar AWS IoT Greengrass, tiene que crear un grupo de Greengrass y, a continuación, añadir un núcleo de Greengrass. Para obtener más información acerca de la configuración de AWS IoT Greengrass, consulte Introducción a AWS IoT Greengrass.

Después de configurar AWS y AWS IoT Greengrass, tiene que configurar algunos permisos adicionales para AWS IoT Greengrass.

Para configurar permisos de AWS IoT Greengrass

1. Vaya a la consola de IAM.
2. En el panel de navegación, elija Roles y, a continuación, busque y seleccione Greengrass_ServiceRole.
3. Elija Attach policies (Asociar políticas), seleccione AmazonS3FullAccess y AWSIoTFullAccess y, a continuación, elija Attach policy (Asociar política).
4. Vaya a la consola de AWS IoT.
5. En el panel de navegación, elija Greengrass, elija Groups (Grupos) y, a continuación, elija el grupo de Greengrass que creó con anterioridad.
6. Elija Settings (Configuración) y, a continuación, elija Add role (Añadir rol).
7. Elija Greengrass_ServiceRole y, a continuación, elija Save (Guardar).

Conecte la placa a AWS IoT y configure la demostración de FreeRTOS.

1. Registro de la placa de MCU con AWS IoT

Después de registrar su placa, tiene que crear y asociar una nueva política de Greengrass al certificado del dispositivo.

Para crear una nueva política de AWS IoT Greengrass

1. Vaya a la consola de AWS IoT.
2. En el panel de navegación, elija Secure (Seguridad), elija Policies (Políticas) y, a continuación, elija Create (Crear).
3. Especifique un nombre que identifique la política.
4. En la sección Añadir declaraciones, elija Modo avanzado. Copie y pegue la siguiente política JSON en la ventana del editor de políticas:

```
{
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": "*"
}
```

   Esta política concede los permisos de AWS IoT Greengrass a todos los recursos.
5. Seleccione Crear.

Para asociar la política de AWS IoT Greengrass al certificado de su dispositivo

1. Vaya a la consola de AWS IoT.
2. En el panel de navegación, elija Manage (Administrar), elija Things (Objetos) y, a continuación, elija el objeto que creó anteriormente.
3. Elija Security (Seguridad) y, a continuación, elija el certificado asociado a su dispositivo.
4. Elija Policies (Políticas), elija Actions (Acciones) y, a continuación, Attach Policy (Asociar política).
5. Encuentre y elija la política de Greengrass que creó anteriormente y, a continuación, elija Attach (Asociar).

2. Descarga de FreeRTOS

> **Note**
>
> Si va a descargar FreeRTOS desde la consola de FreeRTOS, seleccione Conectar a AWS IoT Greengrass- ***Plataforma*** en lugar de Conectar a AWS IoT- ***Plataforma***.

3. Configuración de las demostraciones de FreeRTOS.

   Abra *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` y defina `CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED`.

Después de configurar AWS IoT y AWS IoT Greengrass y, una vez que haya descargado y configurado FreeRTOS, puede compilar, instalar y ejecutar la demostración de Greengrass en su dispositivo. Para configurar el entorno de desarrollo de hardware y software de la placa, siga las instrucciones que se describen en la Guías de introducción específicas de placas.

La demostración de Greengrass publica una serie de mensajes en el núcleo de Greengrass y en el cliente de MQTT de AWS IoT. Para ver los mensajes en el cliente de MQTT de AWS IoT, abra la consola de AWS IoT, elija Prueba y, a continuación, elija el cliente de prueba MQTT y añada una suscripción a `freertos/demos/ggd`.

En el cliente de MQTT, debería ver las siguientes cadenas:

```
Message from Thing to Greengrass Core: Hello world msg #1!
Message from Thing to Greengrass Core: Hello world msg #0!
Message from Thing to Greengrass Core: Address of Greengrass Core
 found! 123456789012.us-west-2.compute.amazonaws.com
```

Uso de una instancia de Amazon EC2

Si trabaja con una instancia de Amazon EC2

1. Busque el DNS público (IPv4) asociado a su instancia de Amazon EC2: vaya a la consola de Amazon EC2 y, en el panel de navegación izquierdo, elija Instancias. Elija su instancia de Amazon EC2 y, a continuación, elija el panel Descripción. Busque la entrada para el DNS público (IPv4) y anótela.

2. Busque la entrada para Grupos de seguridad y elija el grupo de seguridad asociado con la instancia de Amazon EC2.
3. Elija la pestaña Reglas de entrada y, a continuación, elija Editar reglas de entrada y agregue las siguientes reglas.

   Reglas de entrada

Tipo	Protocolo	Rango de puerto	Origen	Descripción: opcional
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
TCP personalizada	TCP	8883	0.0.0.0/0	Comunicaciones MQTT
TCP personalizada	TCP	8883	::/0	Comunicaciones MQTT
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::0/0	-
Todos ICMP: IPv4	ICMP	Todos	0.0.0.0/0	-
Todos ICMP: IPv4	ICMP	Todos	::0/0	-

4. En la consola de AWS IoT, elija Greengrass y, a continuación, Grupos y elija el grupo Greengrass que creó anteriormente. Elija Settings. Cambie la Detección de conexión local a Administrar manualmente la información de conexión.
5. En el panel de navegación, elija Núcleos y, a continuación, seleccione el núcleo del grupo.
6. Elija Conectividad y asegúrese de que solo tiene un punto de enlace principal (elimine el resto) y que no es una dirección IP (porque está sujeta a cambios). La mejor opción es usar el DNS público (IPv4) que anotó en el primer paso.

7. Agregue el objeto de FreeRTOS IoT que creó en el grupo GG.

   a. Elija la flecha hacia atrás para volver a la página del grupo AWS IoT Greengrass. En el panel de navegación, elija Dispositivos y, a continuación, elija Agregar dispositivo.

   b. Elija Seleccionar un objeto de IoT. Elija su dispositivo y luego elija Finalizar.

8. Añada las suscripciones necesarias: en la página Grupo de Greengrass, elija Suscripciones y, a continuación, seleccione Añadir suscripción e introduzca la información que se muestra aquí.

   Suscripciones

Origen	Objetivo	Tema
TIGG1	IoT Cloud (Nube de IoT)	freertos/demos/ggd

   Donde “Fuente” es el nombre asignado al objeto de AWS IoT que se creó en la consola de AWS IoT cuando registró la placa (“TIGG1” en el ejemplo que se muestra aquí).

9. Comience una implementación de su grupo AWS IoT Greengrass y asegúrese de que la implementación tenga éxito. Ahora debería poder ejecutar correctamente la demostración de detección de AWS IoT Greengrass.

AWS IoT Greengrass V2

Compatibilidad con dispositivos AWS IoT Greengrass V2

La compatibilidad de AWS IoT Greengrass V2 para dispositivos cliente es compatible con versiones anteriores con AWS IoT Greengrass V1. Puede conectar dispositivos cliente FreeRTOS a los dispositivos principales V2 sin cambiar el código de la aplicación. Para permitir que los dispositivos cliente se conecten a un dispositivo principal V2, haga lo siguiente.

- Implemente el software Greengrass en el dispositivo principal de Greengrass. Consulte Conexión de dispositivos cliente a dispositivos principales para conectar un dispositivo a AWS IoT Greengrass V2.
- Para retransmitir mensajes (incluidas las funciones de Lambda) entre los dispositivos cliente, el servicio en la nube de AWS IoT Core y los componentes de Greengrass, implemente y configure el componente puente MQTT.
- Implemente el componente detector de IP para detectar automáticamente la información de conectividad o gestione manualmente los puntos de conexión.

- Consulte Interacción con dispositivos AWS IoT locales para obtener más información.

Para obtener más información, consulte la AWS documentación sobre cómo ejecutar aplicaciones de AWS IoT Greengrass V1 en AWS IoT Greengrass V2.

Demostraciones de coreHTTP

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Estas demostraciones pueden ayudarle a saber cómo utilizar la biblioteca coreHTTP.

Temas

- Demostración de la autenticación mutua de coreHTTP
- Demostración de carga básica de coreHTTP en Amazon S3
- Demostración de descarga básica de coreHTTP en S3
- Demostración básica de varios subprocesos de coreHTTP

Demostración de la autenticación mutua de coreHTTP

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

El proyecto de demostración coreHTTP (Autenticación mutua) muestra cómo establecer una conexión con un servidor HTTP mediante TLS con autenticación mutua entre el cliente y el servidor.

Esta demostración utiliza una implementación de interfaz de transporte basada en mbedTLS para establecer una conexión TLS autenticada por el servidor y el cliente, y muestra un flujo de trabajo de respuesta a solicitudes en HTTP.

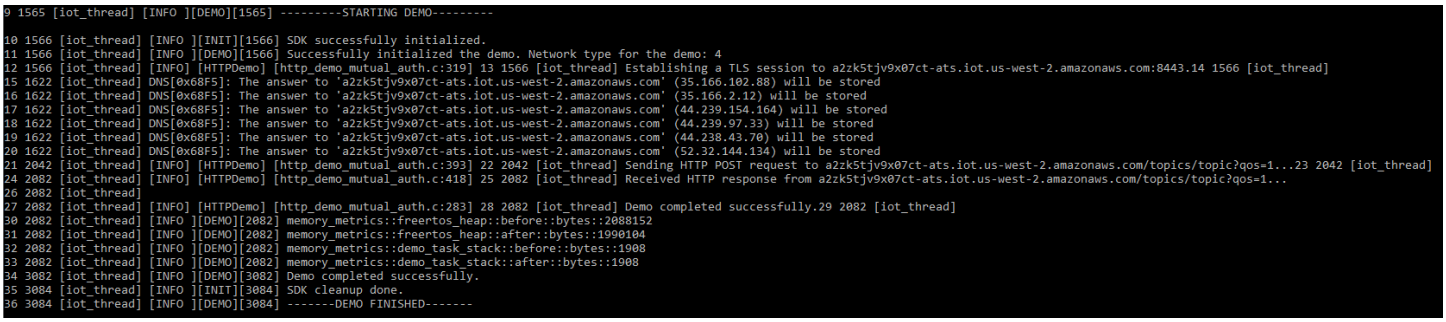
> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Funcionalidad

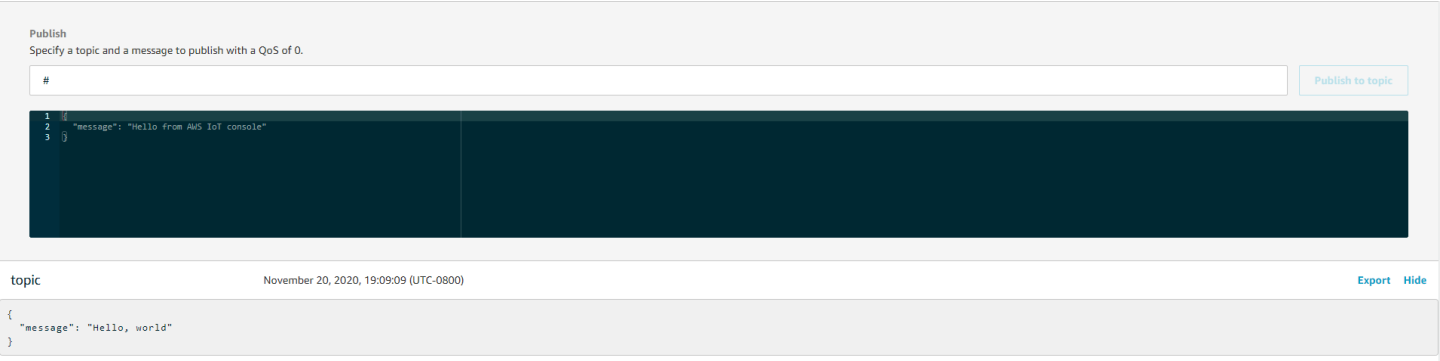
Esta demostración crea una tarea de aplicación única con ejemplos que muestran cómo realizar lo siguiente:

- Conectarse al servidor HTTP en el punto de conexión de AWS IoT.
- Enviar una solicitud POST.
- Recibir la respuesta.
- Desconectarse del servidor.

Tras completar estos pasos, la demostración genera una salida similar a la de la siguiente captura de pantalla.



La consola AWS IoT genera una salida similar a la de la siguiente captura de pantalla.



Organización del código fuente

El archivo fuente de la demostración se llama `http_demo_mutual_auth.c` y se puede encontrar en el directorio *`freertos`*`/demos/coreHTTP/` y en el sitio web de GitHub.

Conexión al servidor HTTP de AWS IoT

La función connectToServerWithBackoffRetries intenta establecer una conexión TLS con autenticación mutua con el servidor HTTP de AWS IoT. Si la conexión falla, se vuelve a intentar cuando se agota el tiempo de espera. El valor de tiempo de espera aumenta exponencialmente hasta que se alcanza el número máximo de intentos o se alcanza el valor de tiempo de espera máximo. La función `RetryUtils_BackoffAndSleep` proporciona valores de tiempo de espera que aumentan exponencialmente y devuelve `RetryUtilsRetriesExhausted` cuando se alcanza el número máximo de intentos. La función `connectToServerWithBackoffRetries` devuelve un estado de error si no se puede establecer la conexión TLS con el agente tras el número de intentos configurado.

Envío de una solicitud HTTP y recepción de la respuesta

La función prvSendHttpRequest muestra cómo enviar una solicitud POST al servidor HTTP de AWS IoT. Para obtener más información sobre cómo realizar una solicitud a la API REST en AWS IoT, consulte Protocolos de comunicación del dispositivo - HTTPS. La respuesta se recibe con la misma llamada a la API coreHTTP, `HTTPClient_Send`.

Demostración de carga básica de coreHTTP en Amazon S3

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto

FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

Este ejemplo muestra cómo enviar una solicitud PUT al servidor HTTP de Amazon Simple Storage Service (Amazon S3) y cargar un archivo pequeño. También realiza una solicitud GET para verificar el tamaño del archivo después de cargarlo. En este ejemplo se utiliza una interfaz de transporte de red que utiliza mbedTLS para establecer una conexión con autenticación mutua entre un cliente de dispositivo IoT que ejecuta coreHTTP y el servidor HTTP de Amazon S3.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Subproceso único frente a varios subprocesos

Hay dos modelos de uso de coreHTTP: subproceso único y varios subprocesos (multitarea). Aunque la demostración de esta sección ejecuta la biblioteca HTTP en un subproceso, en realidad muestra cómo usar coreHTTP en un entorno de un solo subproceso. Solo una de las tareas de esta demostración utiliza la API HTTP. Si bien las aplicaciones con un solo subproceso deben llamar repetidamente a la biblioteca HTTP, las aplicaciones con varios subprocesos pueden enviar solicitudes HTTP en segundo plano dentro de una tarea de agente (o daemon).

Organización del código fuente

El archivo fuente de la demostración se llama `http_demo_s3_upload.c` y se puede encontrar en el directorio *`freertos`*`/demos/coreHTTP/` y en el sitio web de GitHub.

Configuración de la conexión del servidor HTTP de Amazon S3

Esta demostración utiliza una URL prefirmada para conectarse al servidor HTTP de Amazon S3 y autorizar el acceso al objeto que se va a descargar. La conexión TLS del servidor HTTP de Amazon S3 utiliza únicamente la autenticación del servidor. En el nivel de la aplicación, el acceso al objeto se autentica con los parámetros de la consulta de URL prefirmada. Siga los pasos que se indican a continuación para configurar la conexión a AWS.

1. Configure una cuenta de AWS:

   a. Si aún no lo ha hecho, cree una cuenta de AWS.

   b. Las cuentas y los permisos se configuran mediante AWS Identity and Access Management (IAM). Utilice IAM para gestionar los permisos de cada usuario de su cuenta. De forma predeterminada, un usuario no tiene permisos hasta que el propietario raíz los concede.

      i. Para añadir un usuario a su cuenta de AWS, consulte la Guía del usuario de IAM.

      ii. Conceda permiso a su cuenta de AWS para acceder a FreeRTOS y AWS IoT añadiendo esta política:

      - AmazonS3FullAccess

2. Cree un bucket en Amazon S3 siguiendo los pasos que se indican en ¿Cómo se puede crear un bucket de S3? en la Guía del usuario de Amazon Simple Storage Service.
3. Cargue un archivo en Amazon S3 siguiendo los pasos que se indican en ¿Cómo puedo cargar archivos y carpetas en un bucket de S3? .
4. Genere una URL prefirmada mediante el script ubicado en el archivo `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`.

   Para conocer las instrucciones de uso, consulte el archivo `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`.

Funcionalidad

La demostración se conecta primero al servidor HTTP de Amazon S3 con la autenticación del servidor TLS. A continuación, crea una solicitud HTTP para cargar los datos especificados en `democonfigDEMO_HTTP_UPLOAD_DATA`. Después de cargar el archivo, comprueba que el archivo se haya cargado correctamente solicitando el tamaño del archivo. El código fuente de la demostración se encuentra en el sitio web de GitHub.

Conexión al servidor HTTP de Amazon S3

La función connectToServerWithBackoffRetries intenta establecer una conexión TCP con el servidor HTTP. Si la conexión falla, se vuelve a intentar cuando se agota el tiempo de espera. El valor de tiempo de espera aumenta exponencialmente hasta que se alcanza el número máximo de intentos o se alcanza el valor de tiempo de espera máximo. La función

`connectToServerWithBackoffRetries` devuelve un estado de error si no se puede establecer la conexión TCP con el servidor tras el número de intentos configurado.

La función `prvConnectToServer` muestra cómo establecer una conexión con el servidor HTTP de Amazon S3 utilizando únicamente la autenticación del servidor. Utiliza la interfaz de transporte basada en mbedTLS que está implementada en el archivo `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c`. La definición de `prvConnectToServer` se puede encontrar en el sitio web de GitHub.

Carga de datos

La función `prvUploadS3ObjectFile` muestra cómo crear una solicitud PUT y especificar el archivo que se va a cargar. El bucket de Amazon S3 en el que se carga el archivo y el nombre del archivo que se va a cargar se especifican en la URL prefirmada. Para ahorrar memoria, se utiliza el mismo búfer para los encabezados de las solicitudes y para recibir la respuesta. La respuesta se recibe de forma sincrónica mediante la función de la API `HTTPClient_Send`. Se espera un código de estado de respuesta `200 OK` del servidor HTTP de Amazon S3. Cualquier otro código de estado es un error.

El código fuente de `prvUploadS3ObjectFile()` se encuentra en el sitio web de GitHub.

Verificación de la carga

La función `prvVerifyS3ObjectFileSize` realiza llamadas a `prvGetS3ObjectFileSize` para recuperar el tamaño del objeto del bucket de S3. Actualmente, el servidor HTTP de Amazon S3 no admite solicitudes HEAD que utilicen una URL prefirmada, por lo que se solicita el byte 0. El tamaño del archivo se incluye en el campo de encabezado `Content-Range` de la respuesta. Se espera una respuesta `206 Partial Content` del servidor. Cualquier otro código de estado de respuesta es un error.

El código fuente de `prvGetS3ObjectFileSize()` se encuentra en el sitio web de GitHub.

Demostración de descarga básica de coreHTTP en S3

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto

FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

Esta demostración muestra cómo utilizar solicitudes de rango para descargar archivos del servidor HTTP de Amazon S3. Las solicitudes de rango se admiten de forma nativa en la API coreHTTP cuando se utiliza `HTTPClient_AddRangeHeader` para crear la solicitud HTTP. Para un entorno de microcontroladores, se recomienda encarecidamente utilitzar solicitudes de rango. Al descargar un archivo grande en rangos separados, en lugar de hacerlo en una sola solicitud, se puede procesar cada sección del archivo sin bloquear el socket de red. Las solicitudes de rango reducen el riesgo de que se pierdan paquetes, lo que requiere retransmisiones en la conexión TCP, por lo que mejoran el consumo de energía del dispositivo.

En este ejemplo se utiliza una interfaz de transporte de red que utiliza mbedTLS para establecer una conexión con autenticación mutua entre un cliente de dispositivo IoT que ejecuta coreHTTP y el servidor HTTP de Amazon S3.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Subproceso único frente a varios subprocesos

Hay dos modelos de uso de coreHTTP: subproceso único y varios subprocesos (multitarea). Aunque la demostración de esta sección ejecuta la biblioteca HTTP en un subproceso, en realidad muestra cómo usar coreHTTP en un entorno de un solo subproceso (solo una tarea utiliza la API HTTP en la demostración).. Si bien las aplicaciones con un solo subproceso deben llamar repetidamente a la biblioteca HTTP, las aplicaciones con varios subprocesos pueden enviar solicitudes HTTP en segundo plano dentro de una tarea de agente (o daemon).

Organización del código fuente

El proyecto de demostración se llama `http_demo_s3_download.c` y se puede encontrar en el directorio *`freertos`*`/demos/coreHTTP/` y en el sitio web de GitHub.

Configuración de la conexión del servidor HTTP de Amazon S3

Esta demostración utiliza una URL prefirmada para conectarse al servidor HTTP de Amazon S3 y autorizar el acceso al objeto que se va a descargar. La conexión TLS del servidor HTTP de Amazon S3 utiliza únicamente la autenticación del servidor. En el nivel de la aplicación, el acceso al objeto se autentica con los parámetros de la consulta de URL prefirmada. Siga los pasos que se indican a continuación para configurar la conexión a AWS.

1. Configure una cuenta de AWS:
   a. Si aún no lo ha hecho, cree y active una cuenta de AWS.
   b. Las cuentas y los permisos se configuran mediante AWS Identity and Access Management (IAM). IAM le permite gestionar los permisos de cada usuario de su cuenta. De forma predeterminada, un usuario no tiene permisos hasta que el propietario raíz los concede.
      i. Para añadir un usuario a la cuenta de AWS, consulte la Guía de usuario de IAM.
      ii. Conceda permiso a su cuenta de AWS para acceder a FreeRTOS y AWS IoT añadiendo estas políticas:
         • AmazonS3FullAccess
2. Cree un bucket en S3 siguiendo los pasos que se indican en ¿Cómo se puede crear un bucket de S3? en la Guía del usuario de la consola de Amazon Simple Storage Service.
3. Cargue un archivo en S3 siguiendo los pasos que se indican en ¿Cómo puedo cargar archivos y carpetas en un bucket de S3? .
4. Genere una URL prefirmada mediante el script ubicado en `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`. Para obtener instrucciones, consulte `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`.

Funcionalidad

La demostración recupera primero el tamaño del archivo. Luego, solicita cada rango de bytes secuencialmente, en un bucle, con tamaños de rango de `democonfigRANGE_REQUEST_LENGTH`.

El código fuente de la demostración se encuentra en el sitio web de GitHub.

Conexión al servidor HTTP de Amazon S3

La función connectToServerWithBackoffRetries() intenta establecer una conexión TCP con el servidor HTTP. Si la conexión falla, se vuelve a intentar cuando se agota el tiempo de espera. El valor de tiempo de espera aumentará exponencialmente hasta que se alcance el número máximo de intentos o se alcance el valor de tiempo de espera máximo. `connectToServerWithBackoffRetries()` devuelve un estado de error si la conexión TCP con el servidor no se puede establecer después del número de intentos configurado.

La función `prvConnectToServer()` muestra cómo establecer una conexión con el servidor HTTP de Amazon S3 utilizando únicamente la autenticación del servidor. Utiliza la interfaz de transporte basada en mbedTLS que se implementa en el archivo FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c.

El código fuente de `prvConnectToServer()` se puede encontrar en GitHub.

Creación de una solicitud de rango

La función API `HTTPClient_AddRangeHeader()` admite la serialización de un rango de bytes en los encabezados de las solicitudes HTTP para formar una solicitud de rango. En esta demostración, las solicitudes de rango se utilizan para recuperar el tamaño del archivo y solicitar cada sección del archivo.

La función `prvGetS3ObjectFileSize()` recupera el tamaño del archivo del bucket de S3. El encabezado `Connection: keep-alive` se añade en esta primera solicitud a Amazon S3 para mantener la conexión abierta después de enviar la respuesta. Actualmente, el servidor HTTP de S3 no admite solicitudes HEAD que utilicen una URL prefirmada, por lo que se solicita el byte 0. El tamaño del archivo se incluye en el campo de encabezado `Content-Range` de la respuesta. Se espera una respuesta `206 Partial Content` del servidor; cualquier otro código de estado de respuesta recibido es un error.

El código fuente de `prvGetS3ObjectFileSize()` se puede encontrar en GitHub.

Después de recuperar el tamaño del archivo, esta demostración crea una nueva solicitud de rango para cada rango de bytes del archivo que se va a descargar. Utiliza `HTTPClient_AddRangeHeader()` para cada sección del archivo.

Envío de solicitudes de rango y recepción de respuestas

La función `prvDownloadS3ObjectFile()` envía las solicitudes de rango en un bucle hasta que se descarga todo el archivo. La función API `HTTPClient_Send()` envía una solicitud y recibe la

respuesta de forma sincrónica. Cuando la función regresa, la respuesta se recibe en un `xResponse`. A continuación, se comprueba que el código de estado sea `206 Partial Content` y el número de bytes descargados hasta el momento se incrementa en función del valor del encabezado `Content-Length`.

El código fuente de `prvDownloadS3ObjectFile()` se puede encontrar en GitHub.

Demostración básica de varios subprocesos de coreHTTP

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

Esta demostración utiliza las colas seguras para subprocesos de FreeRTOS para almacenar las solicitudes y respuestas en espera de procesarse. En esta demostración, hay tres tareas que hay que tener en cuenta.

- La tarea principal espera a que las solicitudes aparezcan en la cola de solicitudes. Enviará esas solicitudes a través de la red y, a continuación, colocará la respuesta en la cola de respuestas.
- Una tarea de solicitud crea objetos de solicitud de biblioteca HTTP para enviarlos al servidor y los coloca en la cola de solicitudes. Cada objeto de solicitud especifica un rango de bytes del archivo de S3 que la aplicación ha configurado para su descarga.
- Una tarea de respuesta espera a que las respuestas aparezcan en la cola de respuestas. Registra todas las respuestas que recibe.

Esta demostración básica de varios subprocesos está configurada para usar una conexión TLS únicamente con la autenticación del servidor, que es requerida por el servidor HTTP de Amazon S3. La autenticación de la capa de aplicación se realiza mediante los parámetros de la versión 4 de la firma en la consulta de URL prefirmada.

Organización del código fuente

El proyecto de demostración se llama `http_demo_s3_download_multithreaded.c` y se puede encontrar en el directorio *`freertos`*`/demos/coreHTTP/` y en el sitio web de GitHub.

Creación del proyecto de demostración

El proyecto de demostración usa la edición comunitaria gratuita de Visual Studio. Para crear la demostración:

1. Abra el archivo de la solución de Visual Studio `mqtt_multitask_demo.sln` desde el IDE de Visual Studio.
2. Seleccione Crear solución en el menú Crear del IDE.

> **Note**
>
> Si utiliza Microsoft Visual Studio 2017 o una versión anterior, debe seleccionar un conjunto de herramientas de plataforma compatible con su versión: Proyecto -> Propiedades de RTOSDemos -> Conjunto de herramientas de plataforma.

Configuración del proyecto de demostración

La demostración usa la pila FreeRTOS+TCP TCP/IP, así que siga las instrucciones proporcionadas para el proyecto inicial de TCP/IP para:

1. Instalar los componentes necesarios (como WinPCap).
2. Si lo desea, configure una dirección IP estática o dinámica, una dirección de puerta de enlace y una máscara de red.
3. Si lo desea, configure una dirección MAC.
4. Seleccione una interfaz de red Ethernet en su máquina host.
5. Es importante probar la conexión de red antes de intentar ejecutar la demostración de HTTP.

Configuración de la conexión del servidor HTTP de Amazon S3

Siga las instrucciones que se indican en Configuración de la conexión del servidor HTTP de Amazon S3 para la demostración de descarga básica de coreHTTP.

Funcionalidad

La demostración crea tres tareas en total:

- Una que envía solicitudes y recibe respuestas a través de la red.
- Uno que crea solicitudes para enviarlas.
- Uno que procesa las respuestas recibidas.

En esta demostración, la tarea principal:

1. Crea las colas de solicitudes y respuestas.
2. Crea la conexión al servidor.
3. Crea las tareas de solicitudes y respuestas.
4. Espera a que la cola de solicitudes envíe las solicitudes a través de la red.
5. Coloca las respuestas recibidas a través de la red en la cola de respuestas.

La tarea de solicitud:

1. Crea cada una de las solicitudes de rango.

La tarea de respuesta:

1. Uno que procesa las respuestas recibidas.

Typedefs

La demostración define las siguientes estructuras para admitir varios subprocesos.

Elementos de solicitud

Las siguientes estructuras definen un elemento de solicitud para colocarlo en la cola de solicitudes. El elemento de solicitud se copia en la cola después de que la tarea de solicitud cree una solicitud HTTP.

```
/**
 * @brief Data type for the request queue.
```

```
 *
 * Contains the request header struct and its corresponding buffer, to be
 * populated and enqueued by the request task, and read by the main task. The
 * buffer is included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct RequestItem
{
    HTTPRequestHeaders_t xRequestHeaders;
    uint8_t ucHeaderBuffer[ democonfigUSER_BUFFER_LENGTH ];
} RequestItem_t;
```

Elemento de respuesta

Las siguientes estructuras definen un elemento de respuesta para colocarlo en la cola de respuestas. El elemento de respuesta se copia en la cola después de que la tarea HTTP principal reciba una respuesta a través de la red.

```

/**
 * @brief Data type for the response queue.
 *
 * Contains the response data type and its corresponding buffer, to be enqueued
 * by the main task, and interpreted by the response task. The buffer is
 * included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct ResponseItem
{
    HTTPResponse_t xResponse;
    uint8_t ucResponseBuffer[ democonfigUSER_BUFFER_LENGTH ];
} ResponseItem_t;
```

Tarea principal de envío HTTP

La tarea principal de la aplicación:

1. Analiza la URL prefirmada de la dirección del host para establecer una conexión con el servidor HTTP de Amazon S3.
2. Analiza la URL de prefirmada de la ruta de acceso a los objetos del bucket de S3.
3. Se conecta al servidor HTTP de Amazon S3 utilizando TLS con la autenticación del servidor.
4. Crea las colas de solicitudes y respuestas.
5. Crea las tareas de solicitudes y respuestas.

La función `prvHTTPDemoTask()` realiza esta configuración y proporciona el estado de demostración. El código fuente de esta función se puede encontrar en GitHub.

En la función `prvDownloadLoop()`, la tarea principal bloquea y espera las solicitudes de la cola de solicitudes. Cuando recibe una solicitud, la envía mediante la función de la API `HTTPClient_Send()`. Si la función de la API se realiza correctamente, coloca la respuesta en la cola de respuestas.

El código fuente de `prvDownloadLoop()` se puede encontrar en GitHub.

Tarea de solicitud HTTP

La tarea de solicitud se especifica en la función `prvRequestTask`. El código fuente de esta función se puede encontrar en GitHub.

La tarea de solicitud recupera el tamaño del archivo en el bucket de Amazon S3. Esto se hace en la función `prvGetS3ObjectFileSize`. El encabezado “Connection: keep-alive” se añade a esta solicitud a Amazon S3 para mantener la conexión abierta después de enviar la respuesta. Actualmente, el servidor HTTP de Amazon S3 no admite solicitudes HEAD que utilicen una URL prefirmada, por lo que se solicita el byte 0. El tamaño del archivo se incluye en el campo de encabezado `Content-Range` de la respuesta. Se espera una respuesta `206 Partial Content` del servidor; cualquier otro código de estado de respuesta recibido es un error.

El código fuente de `prvGetS3ObjectFileSize` se puede encontrar en GitHub.

Después de recuperar el tamaño del archivo, la tarea de solicitud continúa solicitando cada rango del archivo. Cada solicitud de rango se coloca en la cola de solicitudes para que la tarea principal la envíe. El usuario de la demostración configura los rangos de archivos en la macro `democonfigRANGE_REQUEST_LENGTH`. Las solicitudes de rango se admiten de forma nativa en la API de la biblioteca cliente HTTP mediante la función `HTTPClient_AddRangeHeader`. La función `prvRequestS3ObjectRange` muestra cómo utilizar `HTTPClient_AddRangeHeader()`.

El código fuente de la función `prvRequestS3ObjectRange` se puede encontrar en GitHub.

Tarea de respuesta HTTP

Las tareas de respuesta esperan en la cola de respuestas las respuestas recibidas a través de la red. La tarea principal rellena la cola de respuestas cuando recibe correctamente una respuesta HTTP. Esta tarea procesa las respuestas registrando el código de estado, los encabezados y el cuerpo. En el entorno real, una aplicación puede procesar la respuesta escribiendo el cuerpo de la respuesta en una memoria flash, por ejemplo. Si el código de estado de la respuesta no es `206 partial`

`content`, la tarea notifica a la tarea principal que la demostración debe fallar. La tarea de respuesta se especifica en la función `prvResponseTask`. El código fuente de esta función se puede encontrar en GitHub.

Demostración de la biblioteca de trabajos de AWS IoT

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

La demostración de la biblioteca de trabajos de AWS IoT muestra cómo conectarse al servicio de trabajos de AWS IoT mediante una conexión MQTT, recuperar un trabajo desde AWS IoT y procesarlo en un dispositivo. El proyecto de demostración de trabajos de AWS IoT utiliza el puerto FreeRTOS para Windows, por lo que se puede crear y evaluar con la versión de Visual Studio Community en Windows. No se necesita ningún hardware de microcontrolador. La demostración establece una conexión segura con el agente MQTT de AWS IoT mediante TLS de la misma manera que la Demostración de la autenticación mutua de coreMQTT.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Organización del código fuente

El código de demostración se encuentra en el archivo `jobs_demo.c` y se puede encontrar en el sitio web de GitHub o en el directorio *`freertos`*`/demos/jobs_for_aws/`.

Configuración de la conexión con el agente de MQTT de AWS IoT

En esta demostración, utilizará una conexión MQTT con el agente de MQTT de AWS IoT. Esta conexión se configura de la misma manera que la Demostración de la autenticación mutua de coreMQTT.

Funcionalidad

La demostración muestra el flujo de trabajo utilizado para recibir trabajos desde AWS IoT y procesarlos en un dispositivo. La demostración es interactiva y requiere que cree trabajos mediante la consola de AWS IoT o la AWS Command Line Interface (AWS CLI). Para obtener más información sobre cómo crear un trabajo, consulte create-job en la Referencia del comando de la AWS CLI. La demostración requiere que el documento de trabajo tenga una clave `action` establecida en `print` para imprimir un mensaje en la consola.

Consulte el formato siguiente para este documento de trabajo.

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

Puede utilizar la AWS CLI para crear un trabajo, como en el siguiente ejemplo de comando.

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

La demostración también utiliza un documento de trabajo que contiene la clave `action` establecida en `publish` para volver a publicar el mensaje en un tema. Consulte el formato siguiente para este documento de trabajo.

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

La demostración se repite hasta que recibe un documento de trabajo con la clave `action` establecida en `exit` para salir de la demostración. El formato del documento de trabajo es el siguiente.

```
{
```

```
    "action: "exit"
}
```

Punto de entrada de la demostración de trabajos

El código fuente de la función de punto de entrada de la demostración de trabajos se puede encontrar en GitHub. Esta función realiza las operaciones siguientes:

1. Establece una conexión MQTT mediante las funciones auxiliares de `mqtt_demo_helpers.c`.
2. Se suscribe al tema de MQTT correspondiente a la API `NextJobExecutionChanged` utilizando las funciones auxiliares de `mqtt_demo_helpers.c`. La cadena de temas se ensambla antes, mediante macros definidas por la biblioteca de trabajos de AWS IoT.
3. Publica en el tema de MQTT correspondiente a la API `StartNextPendingJobExecution` utilizando las funciones auxiliares de `mqtt_demo_helpers.c`. La cadena de temas se ensambla antes, mediante macros definidas por la biblioteca de trabajos de AWS IoT.
4. Llama repetidamente a `MQTT_ProcessLoop` para recibir los mensajes entrantes que se entregan a `prvEventCallback` para su procesamiento.
5. Cuando la demostración reciba la acción de salida, cancela la suscripción al tema de MQTT y se desconecta mediante las funciones auxiliares del archivo `mqtt_demo_helpers.c`.

Devolución de llamada para mensajes MQTT recibidos

La función prvEventCallback llama a `Jobs_MatchTopic` desde la biblioteca de trabajos de AWS IoT para clasificar el mensaje MQTT entrante. Si el tipo de mensaje corresponde a un nuevo trabajo, se llama a `prvNextJobHandler()`.

La función prvNextJobHandler y las funciones a las que llama analizan el documento de trabajo a partir del mensaje con formato JSON y ejecutan la acción especificada en el trabajo. La función `prvSendUpdateForJob` es de particular interés.

Envío de una actualización para un trabajo en ejecución

La función prvSendUpdateForJob () llama a `Jobs_Update()` desde la biblioteca de trabajos para rellenar la cadena de temas utilizada en la operación de publicación en MQTT que sigue inmediatamente.

Demostraciones de coreMQTT

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Estas demostraciones pueden ayudarle a saber cómo utilizar la biblioteca coreHTTP.

Temas

- Demostración de la autenticación mutua de coreMQTT
- Demostración de conexión compartida del agente coreMQTT

Demostración de la autenticación mutua de coreMQTT

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

El proyecto de demostración de autenticación mutua de coreHTTP muestra cómo establecer una conexión con un agente HTTP mediante TLS con autenticación mutua entre el cliente y el servidor. Esta demostración utiliza una implementación de interfaz de transporte basada en mbedTLS para establecer una conexión TLS autenticada por el servidor y el cliente, y muestra el flujo de trabajo de suscripción-publicación de MQTT en el nivel de QoS 1. Se suscribe a un filtro de temas, luego publica los temas que coinciden con el filtro y espera a que el servidor reciba esos mensajes en el nivel de QoS 1. Este ciclo de publicación en el agente y recepción del mismo mensaje por parte del agente se repite indefinidamente. Los mensajes de esta demostración se envían en QoS 1, lo que garantiza al menos una entrega de acuerdo con la especificación MQTT.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

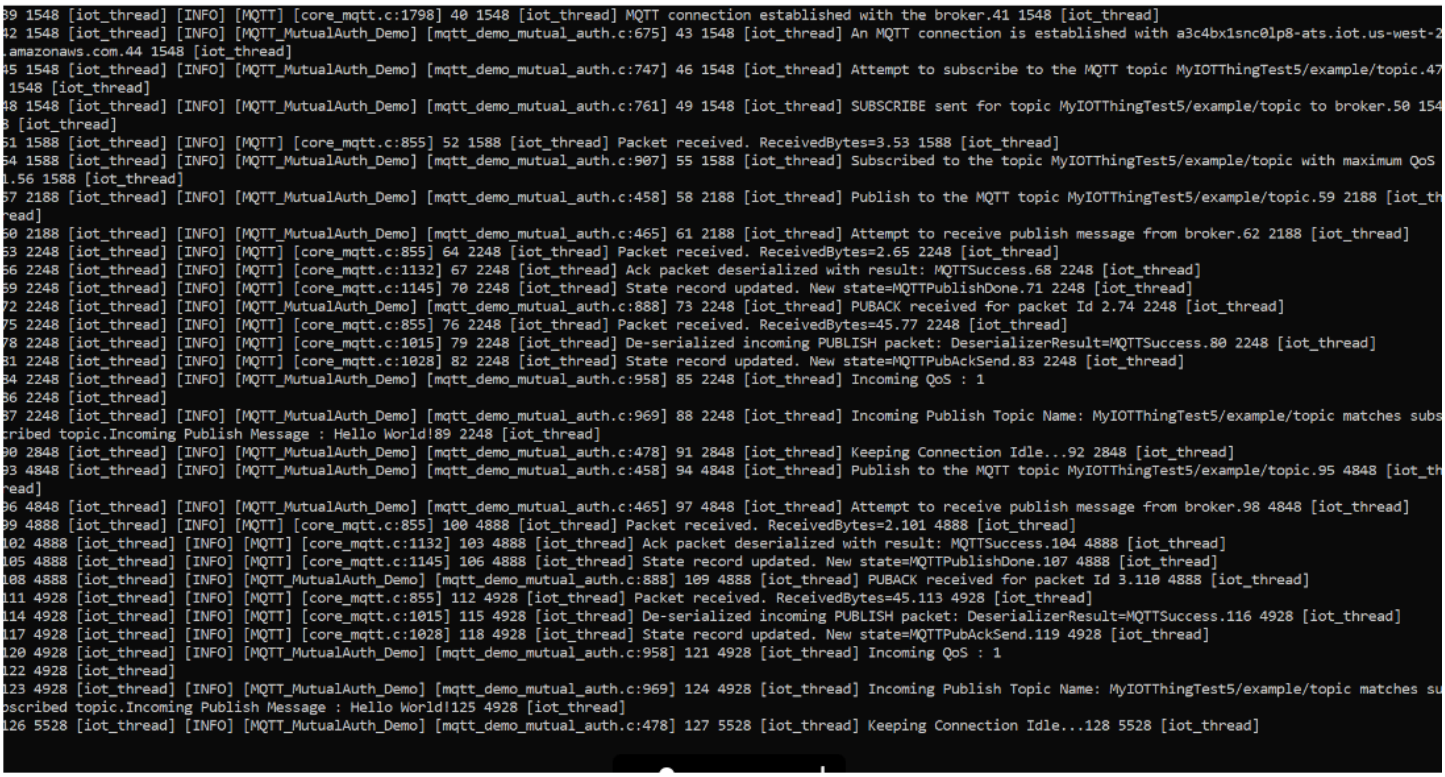
Código fuente

El archivo fuente de la demostración se llama `mqtt_demo_mutual_auth.c` y se puede encontrar en el directorio *`freertos`*`/demos/coreMQTT/` y en el sitio web de GitHub.

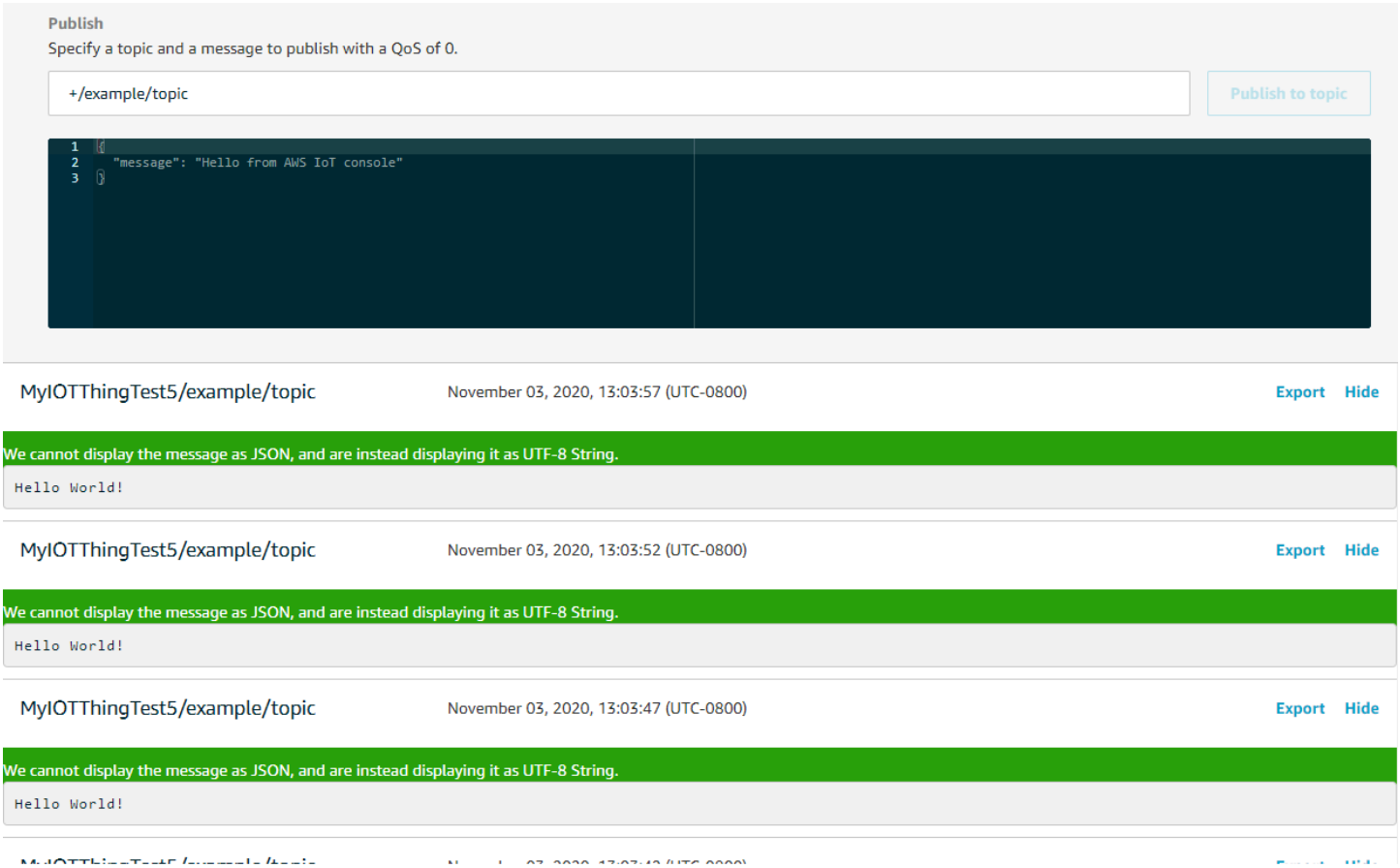
Funcionalidad

La demostración crea una única tarea de aplicación que recorre un conjunto de ejemplos que muestran cómo conectarse al agente, suscribirse a un tema en el agente, publicar en un tema en el agente y, por último, desconectarse del agente. La aplicación de demostración se suscribe y publica en el mismo tema. Cada vez que la demostración publica un mensaje en el agente de MQTT, este envía el mismo mensaje a la aplicación de demostración.

Si se completa correctamente la demostración, se generará una salida similar a la de la siguiente imagen.



La consola de AWS IoT generará una salida similar a la de la siguiente imagen.



Reintento de la lógica con retroceso exponencial y fluctuación

La función prvBackOffForRetry muestra cómo se pueden reintentar las operaciones de red fallidas con el servidor, por ejemplo, las conexiones TLS o las solicitudes de suscripción a MQTT, con retrocesos exponenciales y fluctuaciones. La función calcula el período de retroceso para el siguiente reintento y realiza el retroceso si no se han agotado los reintentos. Como el cálculo del período de retroceso requiere la generación de un número aleatorio, la función utiliza el módulo PKCS11 para generar el número aleatorio. El uso del módulo PKCS11 permite acceder a un generador de números aleatorios verdaderos (TRNG) si la plataforma del proveedor lo admite. Le recomendamos que añada al generador de números aleatorios una fuente de entropía específica del dispositivo para reducir la probabilidad de colisiones entre los dispositivos durante los reintentos de conexión.

Conexión al agente de MQTT

La función prvConnectToServerWithBackOffreTries intenta establecer una conexión TLS con autenticación mutua con el agente de MQTT. Si la conexión falla, se vuelve a intentar tras un período de retroceso. El valor de retroceso aumenta exponencialmente hasta que se alcanza el

número máximo de intentos o se alcanza el valor de período de retroceso máximo. La función `BackoffAlgorithm_GetNextBackoff` proporciona un valor de retroceso que aumenta exponencialmente y devuelve `RetryUtilsRetriesExhausted` cuando se alcanza el número máximo de intentos. La función `prvConnectToServerWithBackoffRetries` devuelve un estado de error si no se puede establecer la conexión TLS con el agente tras el número de intentos configurado.

La función prvCreateMQttConnectionWithBroker muestra cómo establecer una conexión MQTT con un agente de MQTT con una sesión limpia. Utiliza la interfaz de transporte TLS, que está implementada en el archivo `FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c`. Tenga en cuenta que estamos configurando los segundos de permanencia activa para el agente . `xConnectInfo`

La siguiente función muestra cómo se configuran la interfaz de transporte TLS y la función de tiempo en un contexto MQTT mediante la función `MQTT_Init`. También muestra cómo se configura un puntero de la función de devolución de llamada de eventos (`prvEventCallback`). Esta devolución de llamada se utiliza notificar los mensajes entrantes.

Suscripción a un tema de MQTT

La función prvMQTTSubscribeWithBackoffRetries muestra cómo suscribirse a un filtro de temas en el agente de MQTT. El ejemplo muestra cómo suscribirse a un filtro de temas, pero es posible transferir una lista de filtros de temas en la misma llamada a la API de suscripción para suscribirse a más de un filtro de temas. Además, en caso de que el agente de MQTT rechace la solicitud de suscripción, la suscripción volverá a intentarlo, con un retroceso exponencial, durante el valor establecido en `RETRY_MAX_ATTEMPTS`.

Publicación de un tema

La función prvMQTTPublishToTopic muestra cómo publicar en un tema en el agente de MQTT.

Recepción de mensajes entrantes

La aplicación registra una función de devolución de llamada de eventos antes de conectarse al agente, tal y como se ha descrito anteriormente. La función `prvMQTTDemoTask` llama a la función `MQTT_ProcessLoop` para recibir los mensajes entrantes. Cuando se recibe un mensaje MQTT entrante, llama a la función de devolución de llamada del evento registrada por la aplicación. La función prvEventCallback es un ejemplo de dicha función de devolución de llamada de eventos. `prvEventCallback` examina el tipo de paquete entrante y llama al controlador correspondiente. En

el ejemplo siguiente, la función llama a `prvMQTTProcessIncomingPublish()` para gestionar los mensajes de publicación entrantes o a `prvMQTTProcessResponse()` para gestionar los acuses de recibo (ACK).

Proceso de paquetes de publicación MQTT entrantes

La función prvMQTTProcessIncomingPublish muestra cómo procesar un paquete de publicación del agente de MQTT.

Cancelación de la suscripción a un tema

El último paso del flujo de trabajo consiste en cancelar la suscripción al tema para que el agente no envíe ningún mensaje publicado desde `mqttexampleTOPIC`. Esta es la definición de la función prvMQTTUnsubscribeFromTopic.

Cambio de la CA raíz utilizada en la demostración

De forma predeterminada, las demostraciones de FreeRTOS utilizan el certificado CA raíz 1 de Amazon (clave RSA de 2048 bits) para autenticarse en el servidor de AWS IoT Core. Es posible utilizar otros certificados CA para la autenticación del servidor, incluido el certificado CA raíz 3 de Amazon (clave ECC de 256 bits). Para cambiar la CA raíz de la demostración de autenticación mutua coreMQTT:

1. Abra el archivo *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/mqtt_demo_mutual_auth_config.h` en un editor de texto.
2. En el archivo, localice la línea siguiente.

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   Elimine el comentario de esta línea y, si es necesario, muévala más allá del extremo del bloque de comentarios `*/`.
3. Copie el certificado CA que desee utilizar y péguelo en el texto `"...insert here..."`. El resultado debe ser similar al siguiente ejemplo:

   ```
   #define democonfigROOT_CA_PEM   "-----BEGIN CERTIFICATE-----\n"\
   "MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQQDAjA5\n"\
   "MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"\
   "Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"\
   "A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"\
   "Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG8lKl\n"\
   ```

```
"ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVOho6AF2hiRVd9RFgdszflZwjrZt6j\n"\
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"\
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkjOPQQDAgNJADBGAiEA4IWSoxe3jfkr\n"\
"BqWTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyzlvnrxir4tiz+OpAUFteM\n"\
"YyRIHN8wfdVoOw==\n"\
"-----END CERTIFICATE-----\n"
```

4. (Opcional) Puede cambiar la CA raíz para otras demostraciones. Repita los pasos 1 a 3 para cada archivo *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h.

Demostración de conexión compartida del agente coreMQTT

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

El proyecto de demostración de conexión compartida de coreHTTP muestra cómo utilizar una aplicación de varios subprocesos para establecer una conexión con el agente MQTT de AWS utilizando TLS con autenticación mutua entre el cliente y el servidor. Esta demostración utiliza una implementación de interfaz de transporte basada en mbedTLS para establecer una conexión TLS autenticada por el servidor y el cliente, y muestra el flujo de trabajo de suscripción-publicación de MQTT en el nivel de QoS 1. La demostración se suscribe a un filtro de temas, publica los temas que coinciden con el filtro y espera a que el servidor reciba esos mensajes en el nivel de QoS 1. Este ciclo de publicación al agente y de recepción del mismo mensaje de vuelta del agente se repite varias veces para cada tarea creada. Los mensajes de esta demostración se envían en QoS 1, lo que garantiza al menos una entrega de acuerdo con la especificación MQTT.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Esta demostración utiliza una cola segura para subprocesos para almacenar los comandos que interactúan con la API MQTT. En esta demostración, hay dos tareas que hay que tener en cuenta.

- Una tarea del agente MQTT (principal) procesa los comandos de la cola de comandos mientras que otras tareas los ponen en cola. Esta tarea entra en un bucle durante el cual procesa los comandos de la cola de comandos. Si se recibe un comando de terminación, esta tarea saldrá del bucle.
- Una tarea de subpublicación de demostración crea una suscripción a un tema de MQTT, luego crea operaciones de publicación y las envía a la cola de comandos. A continuación, la tarea del agente MQTT ejecuta estas operaciones de publicación. La tarea subpublicación de demostración espera a que finalice la publicación, indicada por la ejecución de la devolución de llamada de finalización del comando y luego introduce un breve retardo antes de iniciar la siguiente publicación. Esta tarea muestra ejemplos de cómo las tareas de aplicación utilizarían la API del agente CoreMQTT.

Para los mensajes de publicación entrantes, el agente de coreMQTT invoca una única función de devolución de llamada. Esta demostración también incluye un administrador de suscripciones que permite a las tareas especificar una devolución de llamada para invocar los mensajes de publicación entrantes sobre temas a los que se han suscrito. En esta demostración, la devolución de llamada de publicación entrante del agente invoca al administrador de suscripciones que distribuya las publicaciones hacia cualquier tarea que haya registrado una suscripción.

Esta demostración utiliza una conexión TLS con autenticación mutua para conectarse. AWS Si la red se desconecta inesperadamente durante la demostración, el cliente intenta volver a conectarse mediante una lógica de retroceso exponencial. Si el cliente se vuelve a conectar correctamente, pero el agente no puede reanudar la sesión anterior, el cliente volverá a suscribirse a los mismos temas que en la sesión anterior.

Un único subproceso frente a varios subprocesos

Hay dos modelos de uso de coreMQTT: subproceso único y varios subprocesos (multitarea). El modelo de subproceso único utiliza la biblioteca coreMQTT únicamente desde un subproceso y requiere que se realicen repetidas llamadas explícitas en la biblioteca MQTT. En cambio, en los casos de uso con varios subprocesos se puede ejecutar el protocolo MQTT en segundo plano dentro de una tarea de agente (o daemon), como se muestra en la demostración documentada aquí. Al ejecutar el protocolo MQTT en una tarea de agente, no es necesario gestionar explícitamente ningún estado de MQTT ni llamar a la función de la API `MQTT_ProcessLoop`. Además, si utiliza una tarea

de agente, varias tareas de la aplicación pueden compartir una única conexión MQTT sin necesidad de primitivas de sincronización, como mutex.

Código fuente

Los archivos fuente de la demostración se llaman `mqtt_agent_task.c` y `simple_sub_pub_demo.c` y se pueden encontrar en el directorio *`freertos`*`/demos/coreMQTT_Agent/` y en el sitio web de GitHub.

Funcionalidad

Esta demostración crea al menos dos tareas: una principal que procesa las solicitudes de llamadas a la API de MQTT y una cantidad configurable de subtareas que crean esas solicitudes. En esta demostración, la tarea principal crea las subtareas, llama al bucle de procesamiento y, después, se limpia. La tarea principal crea una única conexión MQTT con el agente que se comparte entre las subtareas. Las subtareas crean una suscripción a MQTT con el agente y, a continuación, publican los mensajes en ella. Cada subtarea utiliza un tema único para sus publicaciones.

Tarea principal

La tarea principal de la aplicación, RunCoreMQTTAgentDemo, establece una sesión MQTT, crea las subtareas y ejecuta el bucle de procesamiento MQTTAgent_CommandLoop hasta que se recibe un comando de finalización. Si la red se desconecta inesperadamente, la demostración volverá a conectarse al agente en segundo plano y restablecerá las suscripciones con el agente. Una vez finalizado el bucle de procesamiento, se desconecta del agente.

Comandos

Cuando invoca una API de agente coreMQTT, crea un comando que se envía a la cola de tareas del agente, donde se procesa en `MQTTAgent_CommandLoop()`. En el momento en que se crea el comando, se pueden transferir los parámetros opcionales de contexto y devolución de llamada de finalización. Una vez que se complete el comando correspondiente, se invocará la devolución de llamada de finalización con el contexto transferido y cualquier valor devuelto que se haya creado como resultado del comando. La firma de la devolución de llamada de finalización es la siguiente:

```
typedef void (* MQTTAgentCommandCallback_t )( void * pCmdCallbackContext,
                                              MQTTAgentReturnInfo_t * pReturnInfo );
```

El contexto de finalización de comandos lo define el usuario; en esta demostración, es: struct MQTTAgentCommandContext.

Los comandos se consideran completados cuando:

- Se suscribe, cancela la suscripción y publica con QoS > 0: una vez recibido el paquete de acuse de recibo correspondiente.
- Todas las demás operaciones: una vez que se haya invocado la API de coreMQTT correspondiente.

Todas las estructuras utilizadas por el comando, incluida la información de publicación, la información de suscripción y los contextos de finalización, deben permanecer dentro del ámbito de aplicación hasta que se complete el comando. Una tarea de llamada no debe reutilizar ninguna de las estructuras de un comando antes de que se invoque la devolución de llamada de finalización. Tenga en cuenta que, dado que el agente MQTT invoca la devolución de llamada de finalización, se ejecutará en el contexto de subproceso de la tarea del agente, no con la tarea que creó el comando. Los mecanismos de comunicación entre procesos, como las notificaciones o las colas de tareas, se pueden utilizar para indicar la tarea de llamada cuando se ha completado el comando.

Ejecución del bucle de comandos

Los comandos se procesan de forma continua en `MQTTAgent_CommandLoop()`. Si no hay ningún comando que procesar, el bucle esperará el valor máximo establecido en `MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME` para añadir uno a la cola y, si no se añade ningún comando, ejecutará una sola iteración de `MQTT_ProcessLoop()`. Esto garantiza que MQTT Keep-Alive esté gestionado y que todas las publicaciones entrantes se reciban incluso cuando no haya ningún comando en la cola.

La función de bucle de comandos regresará por las siguientes razones:

- Un comando devuelve cualquier código de estado que no sea `MQTTSuccess`. El bucle de comandos devuelve el estado del error, por lo que puede decidir cómo gestionarlo. En esta demostración, se restablece la conexión TCP y se intenta volver a conectarla. Si se produce algún error, se puede volver a conectar en segundo plano sin que intervengan otras tareas que utilicen MQTT.
- Se procesa un comando de desconexión (desde `MQTTAgent_Disconnect`). El bucle de comandos se cierra para que se pueda desconectar el TCP.
- Se procesa un comando de terminación (desde `MQTTAgent_Terminate`). Este comando también marca como error cualquier comando que aún esté en la cola o en espera de un paquete de acuse de recibo, con un código de devolución de `MQTTRecvFailed`.

Administrador de suscripciones

Como en la demostración se utilizan varios temas, un administrador de suscripciones es una forma cómoda de asociar los temas suscritos a devoluciones de llamadas o tareas únicas. El administrador de suscripciones de esta demostración es de un solo subproceso, por lo que no debe utilizarse en varias tareas al mismo tiempo. En esta demostración, solo se llama a las funciones del administrador de suscripciones desde las funciones de devolución de llamada que se transfieren al agente MQTT y se ejecutan únicamente en el contexto del subproceso de la tarea del agente.

Tarea sencilla de suscripción y publicación

Cada instancia de prvSimpleSubscribePublishTask crea una suscripción a un tema de MQTT y crea operaciones de publicación para ese tema. Para demostrar varios tipos de publicación, las tareas con números pares utilizan QoS 0 (que se completan una vez que se envía el paquete de publicación) y las tareas impares utilizan QoS 1 (que se completan al recibir un paquete PUBACK).

Aplicación de demostración de actualizaciones transparentes

FreeRTOS incluye una aplicación de demostración que ilustra la funcionalidad de la biblioteca de actualización inalámbrica (OTA). La aplicación de demostración deOTA se encuentra en el archivo *freertos*/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c o *freertos*/demos/ota/ota_demo_core_http/ota_demo_core_http.c.

La aplicación de demostración de OTA hace lo siguiente:

1. Inicializa la pila de red de FreeRTOS y el grupo de búfer de MQTT.
2. Crea una tarea para probar la biblioteca de OTA con `vRunOTAUpdateDemo()`.
3. Crea un cliente de MQTT mediante `_establishMqttConnection()`.
4. Se conecta al agente de AWS IoT MQTT mediante `IotMqtt_Connect()` y registra una devolución de llamada de desconexión de MQTT: `prvNetworkDisconnectCallback`.
5. Llama a `OTA_AgentInit()` para crear la tarea de OTA y registra una devolución de llamada que se utilizará cuando se haya completado la tarea de OTA.
6. Reutiliza la conexión MQTT con `xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. Si MQTT se desconecta, la aplicación suspende el agente OTA, intenta volver a conectarse utilizando un retroceso exponencial con fluctuación y, a continuación, reanuda el agente OTA.

Antes de poder utilizar las actualizaciones OTA, complete todos los requisitos previos de Actualizaciones vía inalámbrica de FreeRTOS

Después de completar la configuración para las actualizaciones de OTA, descargue, compile, instale y ejecute la demostración de OTA de FreeRTOS en una plataforma que admita la funcionalidad de OTA. Las instrucciones de demostración específicas de dispositivo están disponibles para los siguientes dispositivos calificados por FreeRTOS:

- CC3220SF-LAUNCHXL de Texas Instruments
- Curiosity PIC32MZEF de Microchip
- Espressif ESP32
- Descarga, creación, instalación y ejecución de la demostración de OTA de FreeRTOS en Renesas RX65N

Después de compilar, instalar y ejecutar la aplicación de demostración de OTA en su dispositivo, puede utilizar la consola de AWS IoT o la AWS CLI para crear un trabajo de actualización de OTA. Una vez que haya creado un trabajo de actualización de OTA, conecte un emulador de terminal para ver el progreso de la actualización de OTA. Anote los errores generados durante el proceso.

Un trabajo de actualización de OTA correcto muestra una salida similar a la siguiente. Se han eliminado algunas líneas de la lista en este ejemplo para abreviar.

```
    249 21207 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    250 21247 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=601.
    251 21247 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
 PUBLISH packet: DeserializerResult=MQTTSuccess.
    252 21248 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
 New state=MQTTPubAckSend.
    253 21249 [MQTT Agent Task] [ota_demo_core_mqtt.c:976] [INFO] [MQTT] Received job
 message callback, size 548.
    254 21252 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    255 21253 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    256 21255 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
```

```
    257 21256 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
    258 21257 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    259 21258 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    260 21259 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    261 21260 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-
sha256-ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    262 21261 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    263 21262 [OTA Agent Task] [ota.c:2199] [INFO] [OTA] Job document was accepted.
 Attempting to begin the update.
    264 21263 [OTA Agent Task] [ota.c:2323] [INFO] [OTA] Job parsing success:
 OtaJobParseErr_t=OtaJobParseErrNone, Job name=AFR_OTA-9702f1a3-b747-4c3e-a0eb-
a3b0cf83ddbb
    265 21318 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    266 21418 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    267 21469 [OTA Agent Task] [ota.c:938] [INFO] [OTA] Setting OTA data interface.
    268 21470 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current State=[CreatingFile],
 Event=[ReceivedJobDocument], New state=[CreatingFile]
    269 21482 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=3.
    270 21483 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED
 to topic $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/data/cbor to bro
    271 21484 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[RequestingFileBlock], Event=[CreateFile], New state=[RequestingFileBlock]
    272 21518 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    273 21532 [MQTT Agent Task] [core_mqtt_agent_command_functions.c:76] [INFO] [MQTT]
 Publishing message to $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-
a18b-441b-b435-4a18d4e7671f/
    274 21534 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/get/cbor
    275 21534 [OTA Agent Task] [ota_mqtt.c:1112] [INFO] [OTA] Published to MQTT
 topic to request the next block: topic=$aws/things/__test_infra_thing71/streams/
AFR_OTA-945d320b-a18b-441b-b435-4a1
    276 21537 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[WaitingForFileBlock], Event=[RequestFileBlock], New state=[WaitingForFileBlock]
```

```
   277 21558 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
ReceivedBytes=4217.
   278 21559 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
PUBLISH packet: DeserializerResult=MQTTSuccess.
   279 21560 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
New state=MQTTPublishDone.
   280 21561 [MQTT Agent Task] [ota_demo_core_mqtt.c:1026] [INFO] [MQTT] Received data
message callback, size 4120.
   281 21563 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
Block index=0, Size=4096
   282 21566 [OTA Agent Task] [ota.c:2683] [INFO] [OTA] Number of blocks remaining:
284

   ... // Output removed for brevity

   3672 42745 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
Block index=284, Size=752
   3673 42747 [OTA Agent Task] [ota.c:2633] [INFO] [OTA] Received final block of the
update.
   (428298) ota_pal: No such certificate file: ecdsa-sha256-signer.crt.pem. Using
certificate in ota_demo_config.h.
   3674 42818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
Queued: 285   Processed: 284   Dropped: 0
   3675 42918 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
Queued: 285   Processed: 284   Dropped: 0

   ... // Output removed for brevity

   3678 43197 [OTA Agent Task] [ota.c:2654] [INFO] [OTA] Received entire update and
validated the signature.
   3685 43215 [OTA Agent Task] [ota_demo_core_mqtt.c:862] [INFO] [MQTT] Received
OtaJobEventActivate callback from OTA Agent.

   ... // Output removed for brevity

   2 39 [iot_thread] [INFO ][DEMO][390] ---------STARTING DEMO---------

   [0;32mI (3633) WIFI: WIFI_EVENT_STA_CONNECTED
   [0;32mI (4373) WIFI: SYSTEM_EVENT_STA_GOT_IP

   ... // Output removed for brevity

   4 351 [sys_evt] [INFO ][DEMO][3510] Connected to WiFi access point, ip address:
255.255.255.0.
```

```
    5 351 [iot_thread] [INFO ][DEMO][3510] Successfully initialized the demo. Network
 type for the demo: 1
    6 351 [iot_thread] [ota_demo_core_mqtt.c:1902] [INFO] [MQTT] OTA over MQTT demo,
 Application version 0.9.1
    7 351 [iot_thread] [ota_demo_core_mqtt.c:1323] [INFO] [MQTT] Creating a TLS
 connection to <endpoint>-ats.iot.us-west-2.amazonaws.com:8883.
    9 718 [iot_thread] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=2.
    10 718 [iot_thread] [core_mqtt_serializer.c:970] [INFO] [MQTT] CONNACK session
 present bit not set.
    11 718 [iot_thread] [core_mqtt_serializer.c:912] [INFO] [MQTT] Connection accepted.

    ... // Output removed for brevity

    17 736 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED to
 topic $aws/things/__test_infra_thing71/jobs/notify-next to broker.
    18 737 [OTA Agent Task] [ota_mqtt.c:381] [INFO] [OTA] Subscribed to MQTT topic:
 $aws/things/__test_infra_thing71/jobs/notify-next
    30 818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    31 819 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    32 820 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.statusDetails.updatedBy: 589824]
    33 822 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    34 823 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
    35 824 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
    36 825 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    37 826 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    38 827 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    39 828 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-sha256-
ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    40 829 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    41 830 [OTA Agent Task] [ota.c:2102] [INFO] [OTA] In self test mode.
    42 830 [OTA Agent Task] [ota.c:1936] [INFO] [OTA] New image has a higher version
 number than the current image: New image version=0.9.1, Previous image version=0.9.0
```

```
    43 832 [OTA Agent Task] [ota.c:2120] [INFO] [OTA] Image version is valid: Begin
 testing file: File ID=0
    53 896 [OTA Agent Task] [ota.c:794] [INFO] [OTA] Beginning self-test.
    62 971 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/jobs/AFR_OTA-9702f1a3-b747-4c3e-
a0eb-a3b0cf83ddbb/update to br63 971 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT]
 De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
    65 973 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
 state=MQTTPublishDone.
    64 973 [OTA Agent Task] [ota_demo_core_mqtt.c:902] [INFO] [MQTT] Successfully
 updated with the new image.
```

Configuraciones de demostración inalámbricas

Las configuraciones de demostración de OTA son opciones de configuración específicas de la demostración que se incluyen en `aws_iot_ota_update_demo.c`. Estas configuraciones son diferentes de las configuraciones de la biblioteca OTA que se proporcionan en el archivo de configuración de la biblioteca OTA.

OTA_DEMO_KEEP_ALIVE_SECONDS

Para el cliente MQTT, esta configuración es el intervalo de tiempo máximo que puede transcurrir entre la finalización de la transmisión de un paquete de control y el inicio del envío del siguiente. En ausencia de un paquete de control, se envía un PINGREQ. El agente debe desconectar un cliente que no envíe un mensaje o un paquete PINGREQ durante una vez y media de este intervalo de mantenimiento activo. Esta configuración debe ajustarse en función de los requisitos de la aplicación.

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

El intervalo base, en segundos, que transcurre antes de volver a intentar la conexión de red. La demostración de OTA intentará volver a conectarse después de este intervalo de tiempo base. El intervalo se duplica después de cada intento fallido. También se añade al intervalo un retraso aleatorio, hasta un máximo de este retraso base.

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

El intervalo máximo, en segundos, que transcurre antes de volver a intentar la conexión de red. El retraso de reconexión se duplica en cada intento fallido, pero solo puede llegar hasta este valor máximo, más una fluctuación del mismo intervalo.

Descargue, compile, instale y ejecute la demostración de OTA de FreeRTOS en CC3220SF-LAUNCHXL de Texas Instruments

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Para descargar FreeRTOS y el código de demostración de OTA

- Puede descargar el código fuente desde el sitio de GitHub en https://github.com/FreeRTOS/FreeRTOS.

Para compilar la aplicación de demostración

1. Siga las instrucciones de Introducción a FreeRTOS para importar el proyecto `aws_demos` a Code Composer Studio, configure su punto de enlace de AWS IoT, su SSID y contraseña wifi y un certificado y clave privada para la placa.
2. Abra *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` y defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Compile la solución y asegúrese de que se compila sin errores.
4. Inicie un emulador de terminal y utilice los siguientes ajustes para conectarse a la placa:
   - Velocidad en baudios: 115 200
   - Bits de datos: 8
   - Paridad: ninguna
   - Bits de parada: 1
5. Ejecute el proyecto en la placa para confirmar que puede conectarse al wifi y al agente de mensajes MQTT de AWS IoT.

Cuando se ejecuta, el emulador de terminal debe mostrar texto como el siguiente:

```
    0 1000 [Tmr Svc] Simple Link task created
    Device came up in Station mode
    1 2534 [Tmr Svc] Write certificate...
    2 5486 [Tmr Svc] [ERROR] Failed to destroy object. PKCS11_PAL_DestroyObject failed.
    3 5486 [Tmr Svc] Write certificate...
    4 5776 [Tmr Svc] Security alert threshold = 15
    5 5776 [Tmr Svc] Current number of alerts = 1
    6 5778 [Tmr Svc] Running Demos.
    7 5779 [iot_thread] [INFO ][DEMO][5779] ---------STARTING DEMO---------
    8 5779 [iot_thread] [INFO ][INIT][5779] SDK successfully initialized.
    Device came up in Station mode
    [WLAN EVENT] STA Connected to the AP: afrlab-pepper , BSSID: 74:83:c2:b4:46:27
    [NETAPP EVENT] IP acquired by the device
    Device has connected to afrlab-pepper
    Device IP Address is 192.168.36.176
    9 8283 [iot_thread] [INFO ][DEMO][8282] Successfully initialized the demo. Network
 type for the demo: 1
    10 8283 [iot_thread] [INFO] OTA over MQTT demo, Application version 0.9.0
    11 8283 [iot_thread] [INFO] Creating a TLS connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com:8883.
    12 8852 [iot_thread] [INFO] Creating an MQTT connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com.
    13 8914 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
    14 8914 [iot_thread] [INFO] CONNACK session present bit not set.
    15 8914 [iot_thread] [INFO] Connection accepted.
    16 8914 [iot_thread] [INFO] Received MQTT CONNACK successfully from broker.
    17 8914 [iot_thread] [INFO] MQTT connection established with the broker.
    18 8915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
    19 8953 [OTA Agent T] [INFO] Current State=[RequestingJob], Event=[Start], New
 state=[RequestingJob]
    20 9008 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=3.
    21 9015 [OTA Agent T] [INFO] SUBSCRIBED to topic $aws/things/__test_infra_thing73/
jobs/notify-next to broker.
    22 9015 [OTA Agent T] [INFO] Subscribed to MQTT topic: $aws/things/
__test_infra_thing73/jobs/notify-next
    23 9504 [MQTT Agent ] [INFO] Publishing message to $aws/things/
__test_infra_thing73/jobs/$next/get.
    24 9535 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=2.
    25 9535 [MQTT Agent ] [INFO] Ack packet deserialized with result: MQTTSuccess.
    26 9536 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    27 9537 [OTA Agent T] [INFO] Sent PUBLISH packet to broker $aws/things/
__test_infra_thing73/jobs/$next/get to broker.
```

```
   28 9537 [OTA Agent T] [WARN] OTA Timer handle NULL for Timerid=0, can't stop.
   29 9537 [OTA Agent T] [INFO] Current State=[WaitingForJob],
Event=[RequestJobDocument], New state=[WaitingForJob]
   30 9539 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=120.
   31 9539 [MQTT Agent ] [INFO] De-serialized incoming PUBLISH packet:
DeserializerResult=MQTTSuccess.
   32 9540 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
   33 9540 [MQTT Agent ] [INFO] Received job message callback, size 62.
   34 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution
   35 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobId
   36 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobDocument
   37 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobDocument.afr_ota
   38 9617 [OTA Agent T] [INFO] Failed job document content
check: Required job document parameter was not extracted:
parameter=execution.jobDocument.afr_ota.protocols
   39 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobDocument.afr_ota.files
   40 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=filesize
   41 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=fileid
   42 9619 [OTA Agent T] [INFO] Failed to parse JSON document as AFR_OTA job:
DocParseErr_t=7
   43 9619 [OTA Agent T] [INFO] No active job available in received job document:
OtaJobParseErr_t=OtaJobParseErrNoActiveJobs
   44 9619 [OTA Agent T] [ERROR] Failed to execute state transition handler: Handler
returned error: OtaErr_t=OtaErrJobParserError
   45 9620 [OTA Agent T] [INFO] Current State=[WaitingForJob],
Event=[ReceivedJobDocument], New state=[CreatingFile]
   46 9915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   47 10915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   48 11915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   49 12915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   50 13915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   51 14915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
```

Descarga, compilación, instalación y ejecución de la demostración de OTA de FreeRTOS en Microchip Curiosity PIC32MZEF

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

> **Note**
>
> De acuerdo con Microchip, eliminaremos Curiosity PIC32MZEF (DM320104) de la rama principal del repositorio de integración de referencias de FreeRTOS y ya no lo incluiremos en las nuevas versiones. Microchip ha publicado un aviso oficial en el que indica que PIC32MZEF (DM320104) ya no se recomienda para los nuevos diseños. Aún se puede acceder a los proyectos y al código fuente de PIC32MZEF a través de las etiquetas de las versiones anteriores. Microchip recomienda a los clientes que utilicen la placa de desarrollo PIC32MZ-EF-2.0 (DM320209) de Curiosity para los nuevos diseños. La plataforma PIC32MZv1 todavía se encuentra en la versión 202012.00 del repositorio de integración de referencias de FreeRTOS. Sin embargo, la plataforma ya no es compatible con la versión 202107.00 de la referencia de FreeRTOS.

Para descargar el código de demostración de OTA de FreeRTOS

- Puede descargar el código fuente desde el sitio de GitHub en https://github.com/FreeRTOS/FreeRTOS.

Para compilar la aplicación de demostración de actualización OTA

1. Siga las instrucciones de Introducción a FreeRTOS para importar el proyecto `aws_demos` a MPLAB X IDE, configure el punto de enlace de AWS IoT el SSID y contraseña wifi y un certificado y clave privada para la placa.
2. Abra el archivo `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` e introduzca su certificado.

```
[] = "your-certificate-key";
```

3. Pegue el contenido de su certificado de firma de código aquí:

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Siguiendo el mismo formato que `aws_clientcredential_keys.h` --, cada línea debe acabar con el carácter de línea nueva ('\n') y estar entrecomillado.

Por ejemplo, el certificado debe tener un aspecto similar al siguiente:

```
"-----BEGIN CERTIFICATE-----\n"
"MIIBXTCCAQOgAwIBAgIJAM4DeybZcTwKMAoGCCqGSM49BAMCMCExHzAdBgNVBAMM\n"
"FnRlc3Rf62lnbmVyQGFtYXpvbi5jb20wHhcNMTcxMTAzMTkxODM1WhcNMTgxMTAz\n"
"MTkxODM2WjAhMR8wHQYDVQBBZZZ0ZXN0X3NpZ25lckBhbWF6b24uY29tMFkwEwYH\n"
"KoZIzj0CAQYIKoZIzj0DAQcDQgAERavZfvwL1X+E4dIF7dbkVMUn4IrJ1CAsFkc8\n"
"gZxPzn683H40XMKltDZPEwr9ng78w9+QYQg7ygnr2stz8yhh06MkMCIwCwYDVR0P\n"
"BAQDAgeAMBMGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIF0R\n"
"r5cb7rEUNtWOvGd05MacrgOABfSoVYvBOK9fP63WAqt5h3BaS123coKSGg84twlq\n"
"TkO/pV/xEmyZmZdV+HxV/OM=\n"
"-----END CERTIFICATE-----\n";
```

4. Instale Python 3 o superior.
5. Instale pyOpenSSL ejecutando `pip install pyopenssl`.
6. Copie el certificado de firma de código en formato .pem en la ruta `demos/ota/bootloader/utility/codesigner_cert_utility/`. Cambie el nombre del archivo del certificado a `aws_ota_codesigner_certificate.pem`.
7. Abra *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` y defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
8. Compile la solución y asegúrese de que se compila sin errores.
9. Inicie un emulador de terminal y utilice los siguientes ajustes para conectarse a la placa:
   - Velocidad en baudios: 115 200
   - Bits de datos: 8

- Paridad: ninguna
- Bits de parada: 1

10. Desconecte el depurador de la placa y ejecute el proyecto en la placa para confirmar que puede conectarse al wifi y al agente de mensajes MQTT de AWS IoT.

Cuando ejecuta el proyecto, MPLAB X IDE debe abrir una ventana de salida. Asegúrese de que la pestaña ICD4 está seleccionada. Debería ver los siguientes datos de salida.

```
Bootloader version 00.09.00
[prvBOOT_Init] Watchdog timer initialized.
[prvBOOT_Init] Crypto initialized.

[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] No application image or magic code present at: 0xbd000000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000

[prvValidateImage] Validating image at Bank : 1
[prvValidateImage] No application image or magic code present at: 0xbd100000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd100000

[prvBOOT_ValidateImages] Booting default image.


>0 36246 [IP-task] vDHCPProcess: offer ac140a0eip
                                                  1 36297 [IP-task] vDHCPProcess: offer
 ac140a0eip
                2 36297 [IP-task]

IP Address: 172.20.10.14
3 36297 [IP-task] Subnet Mask: 255.255.255.240
4 36297 [IP-task] Gateway Address: 172.20.10.1
5 36297 [IP-task] DNS Server Address: 172.20.10.1


6 36299 [OTA] OTA demo version 0.9.2
7 36299 [OTA] Creating MQTT Client...
8 36299 [OTA] Connecting to broker...
9 38673 [OTA] Connected to broker.
10 38793 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/$next/get/accepted
```

```
11 38863 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/notify-next
12 38863 [OTA Task] [OTA_CheckForUpdate] Request #0
13 38964 [OTA] [OTA_AgentInit] Ready.
14 38973 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:devthingota ]
15 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
16 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
17 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
18 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
19 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: files
20 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
21 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
22 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
23 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
24 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
25 38975 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
26 38975 [OTA Task] [prvOTA_Close] Context->0x8003b620
27 38975 [OTA Task] [prvPAL_Abort] Abort - OK
28 39964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
29 40964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
30 41964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
31 42964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
32 43964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
33 44964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
34 45964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
35 46964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
36 47964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
```

El emulador de terminal debe mostrar texto como el siguiente:

```
AWS Validate: no valid signature in descr: 0xbd000000
AWS Validate: no valid signature in descr: 0xbd100000


>AWS Launch:  No Map performed. Running directly from address: 0x9d000020?
AWS Launch:  wait for app at: 0x9d000020
WILC1000: Initializing...
0 0

>[None] Seed for randomizer: 1172751941
1 0 [None] Random numbers: 00004272 00003B34 00000602 00002DE3
Chip ID 1503a0
```

```
[spi_cmd_rsp][356][nmi spi]: Failed cmd response read, bus error...

[spi_read_reg][1086][nmi spi]: Failed cmd response, read reg (0000108c)...

[spi_read_reg][1116]Reset and retry 10 108c

Firmware ver. : 4.2.1

Min driver ver : 4.2.1

Curr driver ver: 4.2.1

WILC1000: Initialization successful!

Start Wi-Fi Connection...
Wi-Fi Connected
2 7219 [IP-task] vDHCPProcess: offer c0a804beip
3 7230 [IP-task] vDHCPProcess: offer c0a804beip
4 7230 [IP-task]

IP Address: 192.168.4.190
5 7230 [IP-task] Subnet Mask: 255.255.240.0
6 7230 [IP-task] Gateway Address: 192.168.0.1
7 7230 [IP-task] DNS Server Address: 208.67.222.222


8 7232 [OTA] OTA demo version 0.9.0
9 7232 [OTA] Creating MQTT Client...
10 7232 [OTA] Connecting to broker...
11 7232 [OTA] Sending command to MQTT task.
12 7232 [MQTT] Received message 10000 from queue.
13 8501 [IP-task] Socket sending wakeup to MQTT task.
14 10207 [MQTT] Received message 0 from queue.
15 10256 [IP-task] Socket sending wakeup to MQTT task.
16 10256 [MQTT] Received message 0 from queue.
17 10256 [MQTT] MQTT Connect was accepted. Connection established.
18 10256 [MQTT] Notifying task.
19 10257 [OTA] Command sent to MQTT task passed.
20 10257 [OTA] Connected to broker.
21 10258 [OTA Task] Sending command to MQTT task.
22 10258 [MQTT] Received message 20000 from queue.
23 10306 [IP-task] Socket sending wakeup to MQTT task.
24 10306 [MQTT] Received message 0 from queue.
```

```
25 10306 [MQTT] MQTT Subscribe was accepted. Subscribed.
26 10306 [MQTT] Notifying task.
27 10307 [OTA Task] Command sent to MQTT task passed.
28 10307 [OTA Task] [OTA] Subscribed to topic: $aws/things/Microchip/jobs/$next/get/
accepted

29 10307 [OTA Task] Sending command to MQTT task.
30 10307 [MQTT] Received message 30000 from queue.
31 10336 [IP-task] Socket sending wakeup to MQTT task.
32 10336 [MQTT] Received message 0 from queue.
33 10336 [MQTT] MQTT Subscribe was accepted. Subscribed.
34 10336 [MQTT] Notifying task.
35 10336 [OTA Task] Command sent to MQTT task passed.
36 10336 [OTA Task] [OTA] Subscribed to topic: $aws/things/Microchip/jobs/notify-next

37 10336 [OTA Task] [OTA] Check For Update #0
38 10336 [OTA Task] Sending command to MQTT task.
39 10336 [MQTT] Received message 40000 from queue.
40 10366 [IP-task] Socket sending wakeup to MQTT task.
41 10366 [MQTT] Received message 0 from queue.
42 10366 [MQTT] MQTT Publish was successful.
43 10366 [MQTT] Notifying task.
44 10366 [OTA Task] Command sent to MQTT task passed.
45 10376 [IP-task] Socket sending wakeup to MQTT task.
46 10376 [MQTT] Received message 0 from queue.
47 10376 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:Microchip ]
48 10376 [OTA Task] [OTA] Missing job parameter: execution
49 10376 [OTA Task] [OTA] Missing job parameter: jobId
50 10376 [OTA Task] [OTA] Missing job parameter: jobDocument
51 10378 [OTA Task] [OTA] Missing job parameter: ts_ota
52 10378 [OTA Task] [OTA] Missing job parameter: files
53 10378 [OTA Task] [OTA] Missing job parameter: streamname
54 10378 [OTA Task] [OTA] Missing job parameter: certfile
55 10378 [OTA Task] [OTA] Missing job parameter: filepath
56 10378 [OTA Task] [OTA] Missing job parameter: filesize
57 10378 [OTA Task] [OTA] Missing job parameter: sig-sha256-ecdsa
58 10378 [OTA Task] [OTA] Missing job parameter: fileid
59 10378 [OTA Task] [OTA] Missing job parameter: attr
60 10378 [OTA Task] [OTA] Returned buffer to MQTT Client.
61 11367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
62 12367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
63 13367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
64 14367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
65 15367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

```
66 16367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

Este salida muestra que Curiosity PIC32MZEF de Microchip puede conectarse a AWS IoT y suscribirse a los temas de MQTT necesarios para las actualizaciones OTA. Se esperan mensajes `Missing job parameter` porque no hay trabajos de actualización OTA pendientes.

Descarga, compilación, instalación y ejecuciónde la demostración de OTA de FreeRTOS en Espressif ESP32

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

1. Descargue el origen de FreeRTOS de GitHub. Consulte el archivo README.md para obtener instrucciones. Cree un proyecto en su IDE que incluya todos los orígenes y bibliotecas necesarios.
2. Siga las instrucciones de Introducción a Espressif para configurar la cadena de herramientas basadas en GCC necesarias.
3. Abra *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` y defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
4. Compile el proyecto de demostración ejecutando `make` en el directorio `vendors/espressif/boards/esp32/aws_demos`. Puede instalar el programa de demostración y verificar su salida ejecutando `make flash monitor`, tal y como se describe en Introducción a Espressif.
5. Antes de ejecutar la demostración de actualización OTA:

   - Abra *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` y defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

- Abra `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` y copie el certificado de firma de código SHA-256/ECDSA en:

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Descarga, creación, instalación y ejecución de la demostración de OTA de FreeRTOS en Renesas RX65N

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

En este capítulo se muestra cómo descargar, crear, instalar y ejecutar la demostración de OTA de FreeRTOS en Renesas RX65N

Temas

- Configuración del entorno operativo
- Limpieza de los recursos de AWS
- Importación, configuración del archivo de encabezado y creación de aws_demos y boot_loader

Configuración del entorno operativo

Para los procedimientos de esta sección se requieren los siguientes entornos:

- IDE: e^2 studio 7.8.0, e^2 studio 2020-07
- Cadenas de herramientas: CCRX Compiler v3.0.1
- Dispositivos de destino: RSKRX65N-2MB
- Depuradores: emulador E^2, E^2 Lite
- Software: Renesas Flash Programmer, Renesas Secure Flash Programmer.exe, Tera Term

Para configurar el hardware

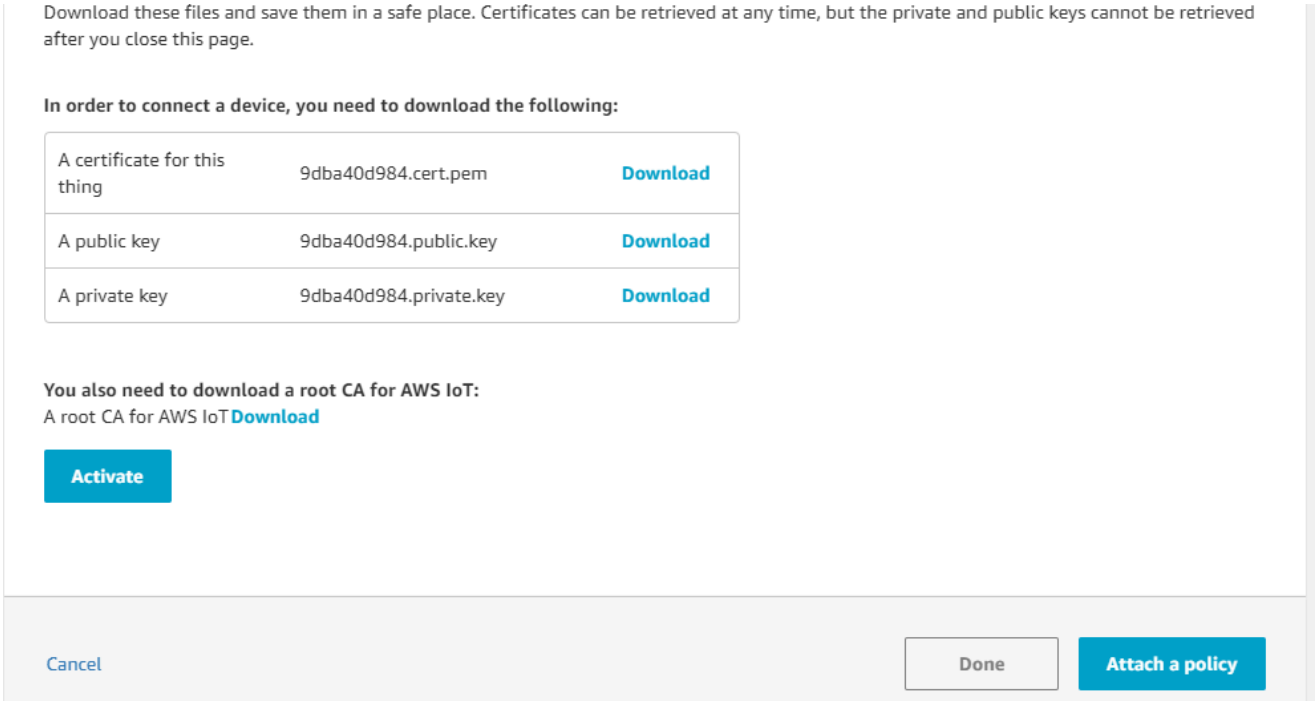
1. Conecte el emulador E^2 Lite y el puerto serie USB a la placa RX65N y al PC.
2. Conecte la fuente de alimentación a RX65N.

Limpieza de los recursos de AWS

1. Para ejecutar las demostraciones de FreeRTOS, debe tener una cuenta de AWS con un usuario de IAM que tenga permiso para acceder a los servicios de AWS IoT. Si aún no lo ha hecho, siga los pasos que se indican en Configuración de su cuenta de AWS y los permisos.
2. Para configurar las actualizaciones de OTA, sigue los pasos que se indican en Requisitos previos de actualización OTA. En concreto, siga los pasos que se indican en Requisitos previos para las actualizaciones de OTA mediante MQTT.
3. Abra la consola de AWS IoT.
4. En el panel de navegación izquierdo, elija Administrar y, a continuación, Objetos.

   Un objeto es una representación de un dispositivo o de una entidad lógica de AWS IoT. Puede ser un dispositivo físico o un sensor (por ejemplo, una bombilla o un interruptor en la pared). También puede ser una entidad lógica, como una instancia de una aplicación o una entidad física que no se conecta con AWS IoT, pero que está relacionada con otros dispositivos que sí lo están (por ejemplo, un automóvil con sensores de motor o un panel de control). AWS IoT proporciona un registro de objetos que le ayuda a gestionarlos.

   a. Elija Crear y después Crear un solo objeto.
   b. Introduzca un Nombre para el objeto y, a continuación, seleccione Siguiente.
   c. Elija Create certificate.
   d. Descargue los tres archivos que se crean y, a continuación, seleccione Activar.
   e. Elija Asociar una política.



f. Seleccione la política que creó en Política de dispositivos.

Cada dispositivo que reciba una actualización OTA utilizando MQTT debe estar registrado como un objeto en AWS IoT y debe tener una política asociada como la que se muestra aquí. Puede encontrar más información acerca de los elementos de los objetos `"Resource"` y `"Action"` en las Acciones de la política principal de AWS IoT y en los recursos de acciones principales de AWS IoT.

Notas

- Los permisos `iot:Connect` permiten que su dispositivo se conecte a AWS IoT a través de MQTT.
- Los permisos `iot:Subscribe` y `iot:Publish` en los temas de trabajos de AWS IoT (`.../jobs/*`) permiten que el dispositivo conectado reciba notificaciones de trabajo y documentos de trabajo, y publique el estado de finalización de una ejecución de trabajo.
- Los permisos `iot:Subscribe` y `iot:Publish` en los temas de flujos de OTA de AWS IoT (`.../streams/*`) permiten que el dispositivo conectado obtenga datos de actualización de OTA desde AWS IoT. Estos permisos son necesarios para realizar actualizaciones de firmware sobre MQTT.
- Los permisos `iot:Receive` permiten a AWS IoT Core publicar mensajes sobre esos temas en el dispositivo conectado. Este permiso se verifica en cada entrega de un

mensaje MQTT. Puede utilizar este permiso para revocar el acceso a los clientes que están actualmente suscritos a un tema.

5. Para crear un perfil de firma de código y registrar un certificado de firma de código en AWS.

   a. Para crear las claves y la certificación, consulte la sección 7.3 “Generación de pares de claves ECDSA-SHA256 con OpenSSL” en la Política de diseño de actualización del firmware de la MCU de Renesas.

   b. Abra la consola de AWS IoT. En el panel de navegación izquierdo, elija Administrar y, a continuación, Trabajos. Seleccione Crear un trabajo y, a continuación, Crear trabajo de actualiz. de OTA.

   c. En Seleccionar dispositivos para actualizar, elija Seleccionar y, a continuación, elija el objeto que creó anteriormente. Seleccione Siguiente.

   d. En la página Crear un trabajo de actualización OTA de FreeRTOS, realice lo siguiente:

      i. En Seleccionar el protocolo para la transferencia de imágenes de firmware, elija MQTT.

      ii. En Seleccionar y firmar la imagen de firmware, elija Firmar una nueva imagen de firmware por mí.

      iii. En Perfil de firma de código, elija Crear.

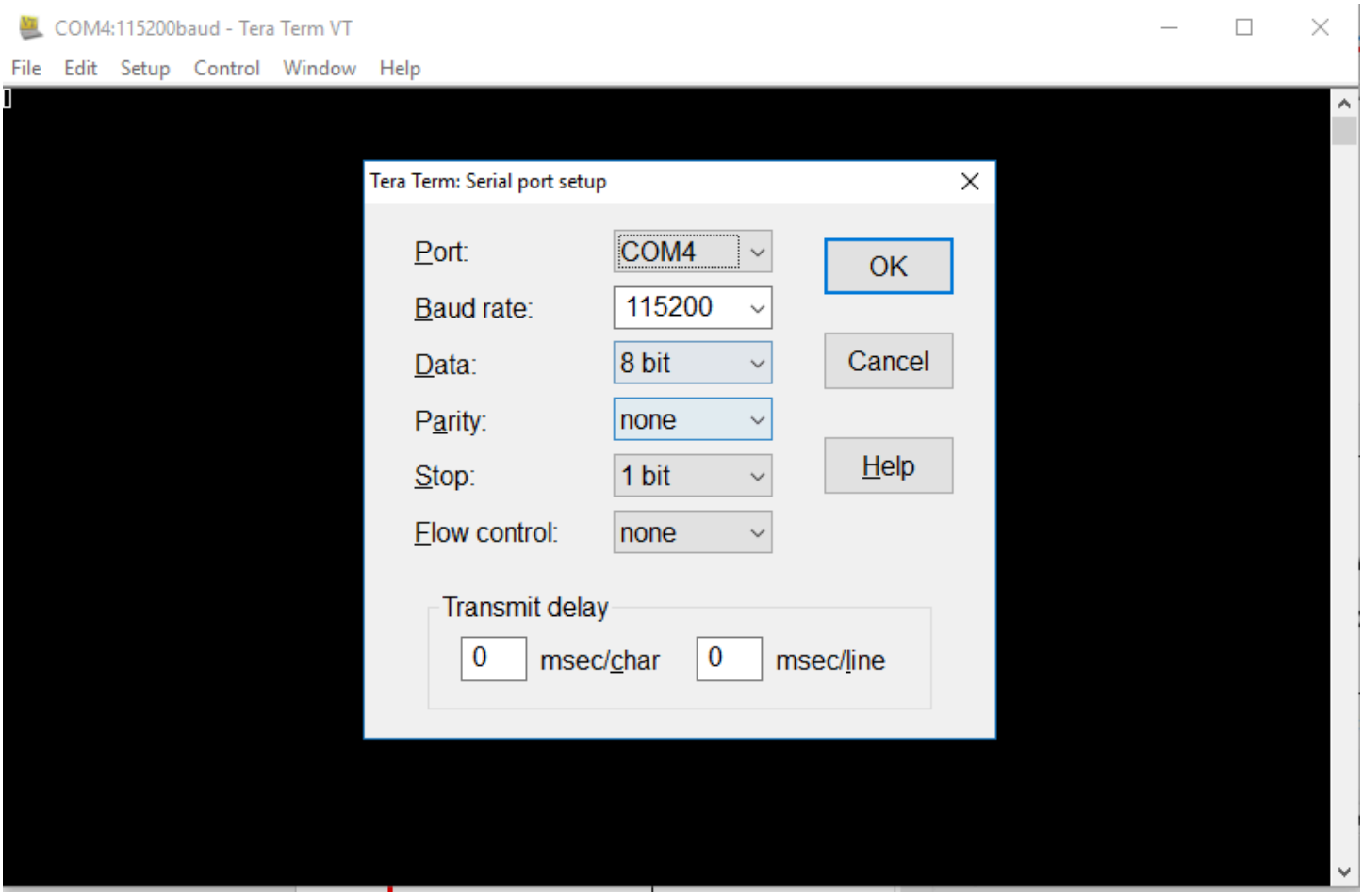
      iv. En la ventana Crear un perfil de firma de código, introduzca un Nombre de perfil. En Plataforma de hardware de dispositivos, seleccione Simulador de Windows. En Certificado de firma de código, elija Importar.

      v. Busque para seleccionar el certificado (`secp256r1.crt`), la clave privada del certificado (`secp256r1.key`) y la cadena de certificados (`ca.crt`).

      vi. Introduzca el Nombre de ruta del certificado de firma de código en el dispositivo. A continuación, elija Create (Crear).

6. Para conceder acceso a la firma de código para AWS IoT, siga los pasos que se indican en Conceder acceso a Code Signing para AWS IoT.

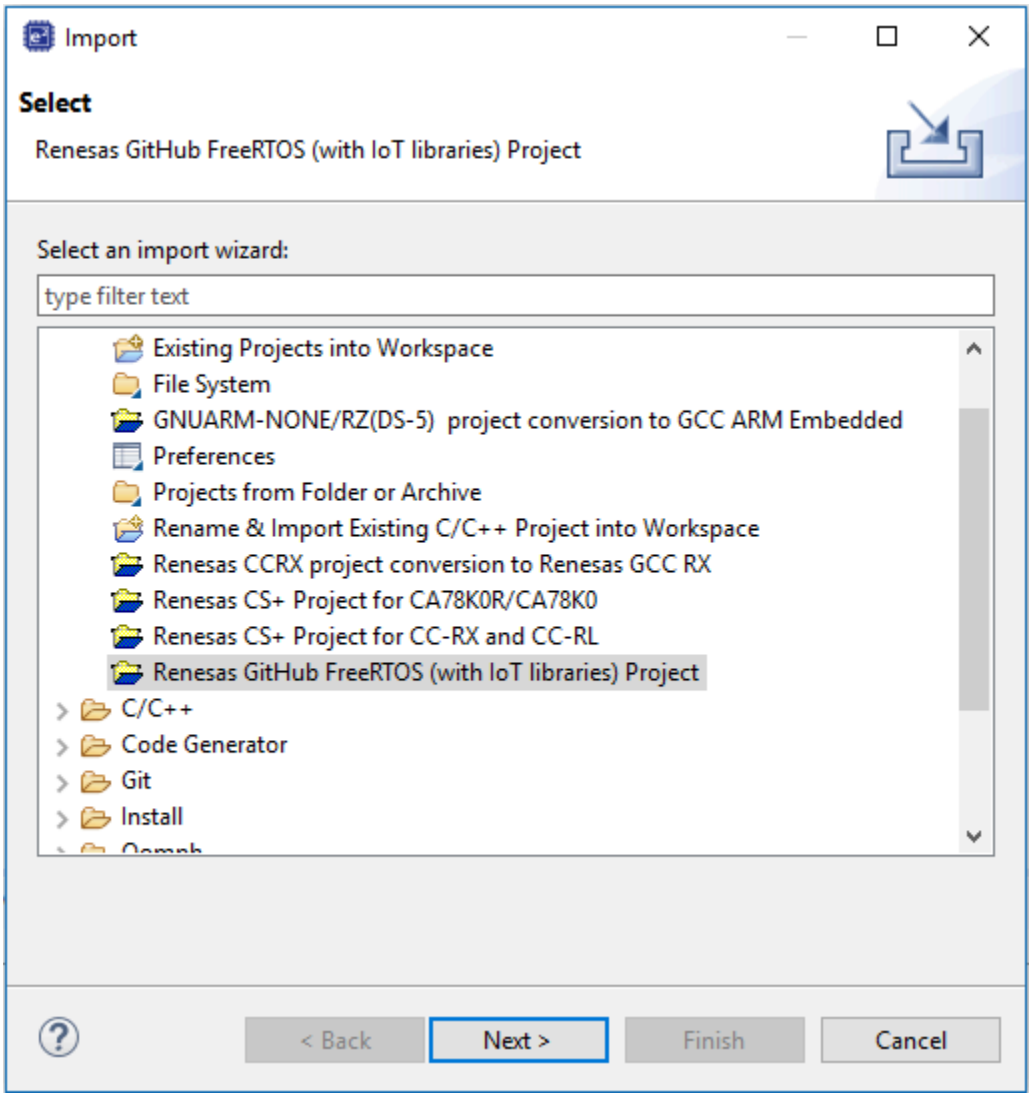
Si no tiene Tera Term instalado en el PC, puede descargarlo desde https://ttssh2.osdn.jp/index.html.en y configurarlo como se muestra aquí. Asegúrese de conectar el puerto serie USB del dispositivo al PC.



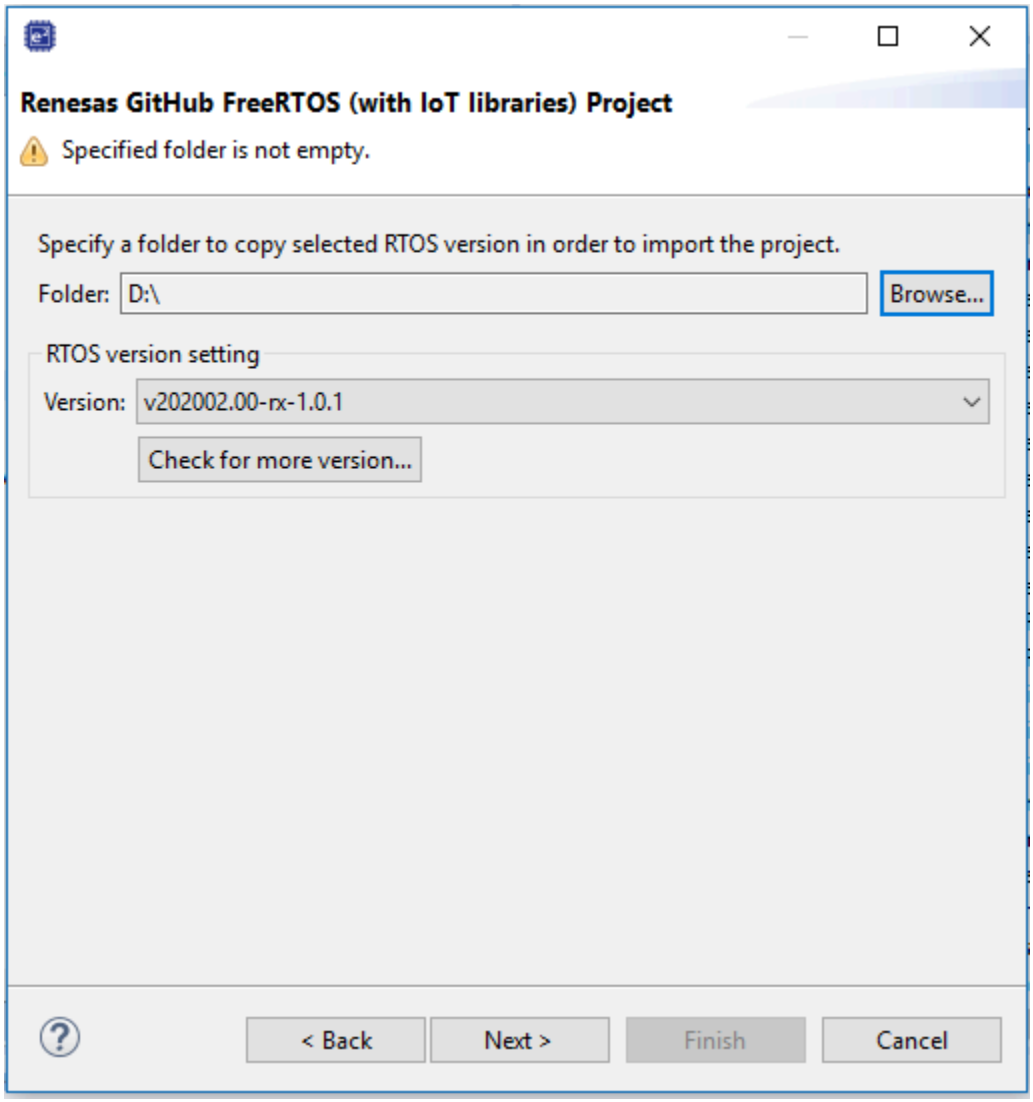
Importación, configuración del archivo de encabezado y creación de aws_demos y boot_loader

Para empezar, seleccione la última versión del paquete FreeRTOS, que se descargará de GitHub y se importará automáticamente al proyecto. De esta forma, puede centrarse en la configuración de FreeRTOS y en escribir el código de la aplicación.

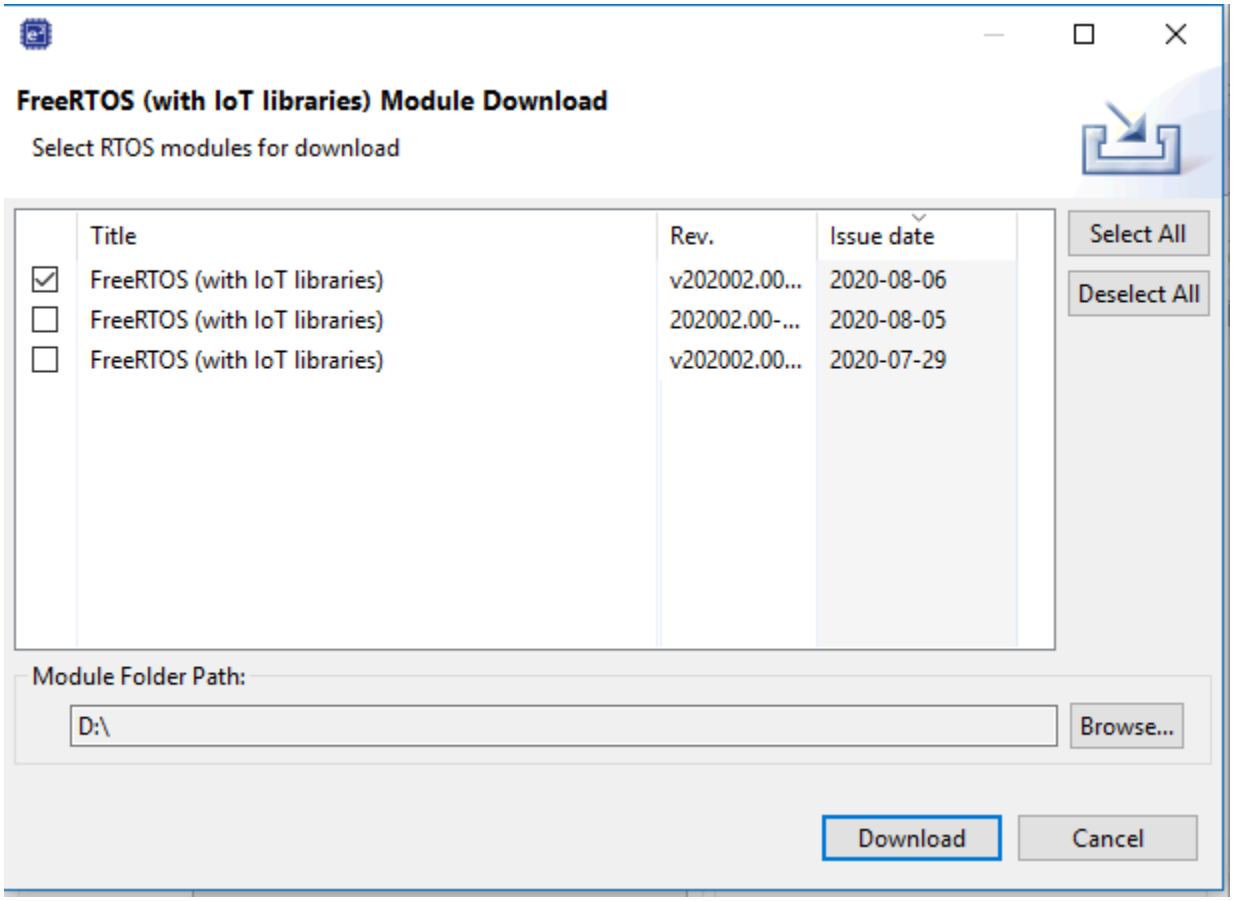
1. Inicie e^2 studio.
2. Elija Archivo y, a continuación, Importar.
3. Seleccione el proyecto Renesas GitHub FreeRTOS (con bibliotecas de IoT).



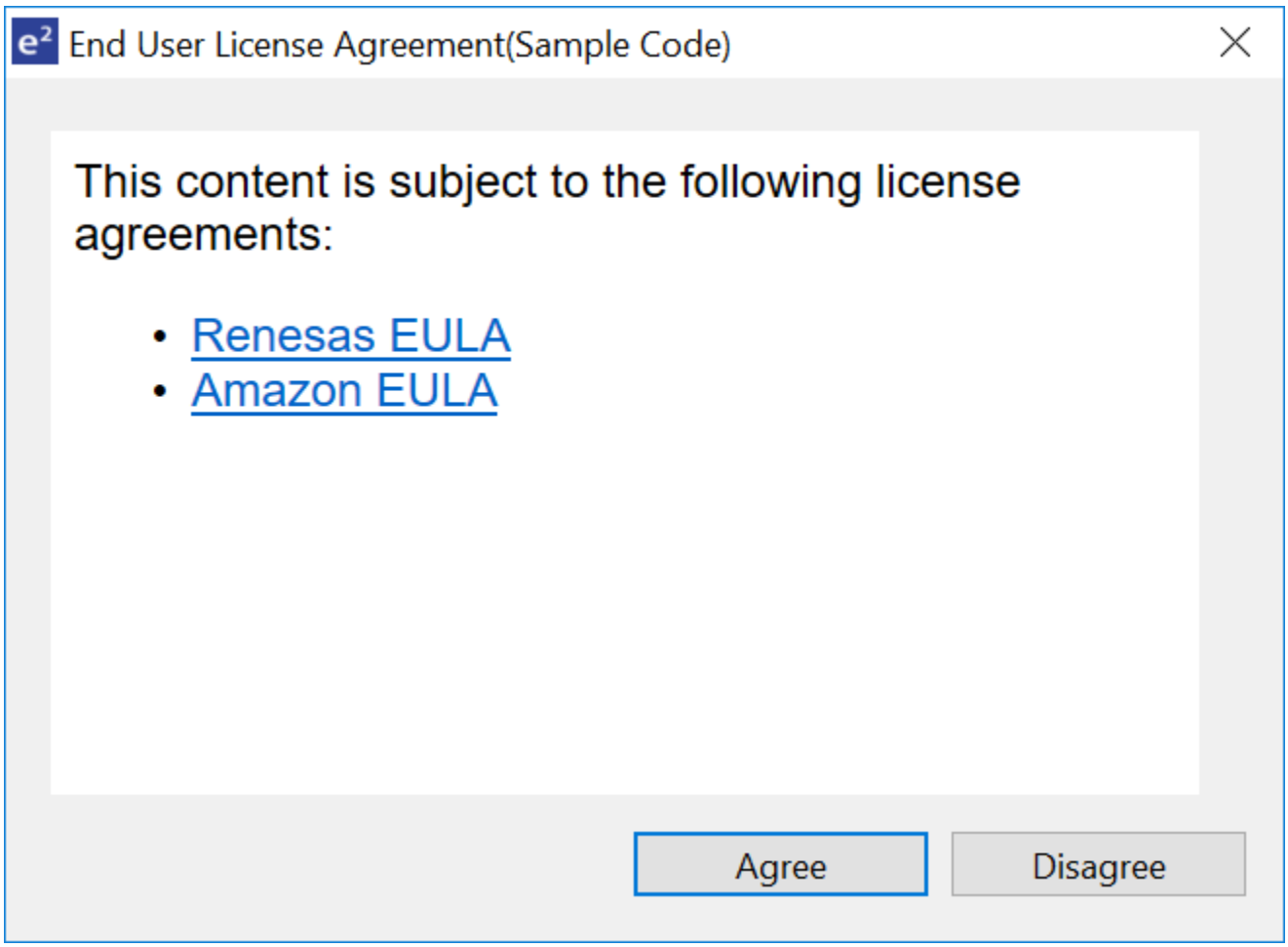
4. Seleccione Buscar más versiones… para mostrar el cuadro de diálogo de descarga.



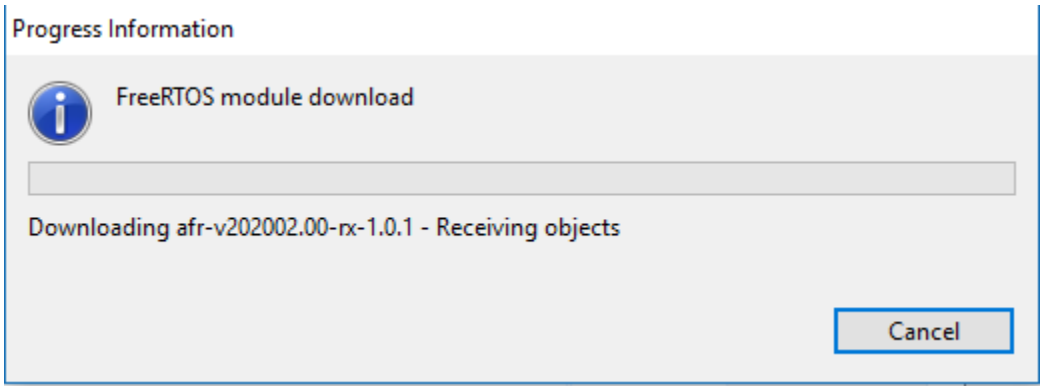
5. Seleccione el paquete más reciente.



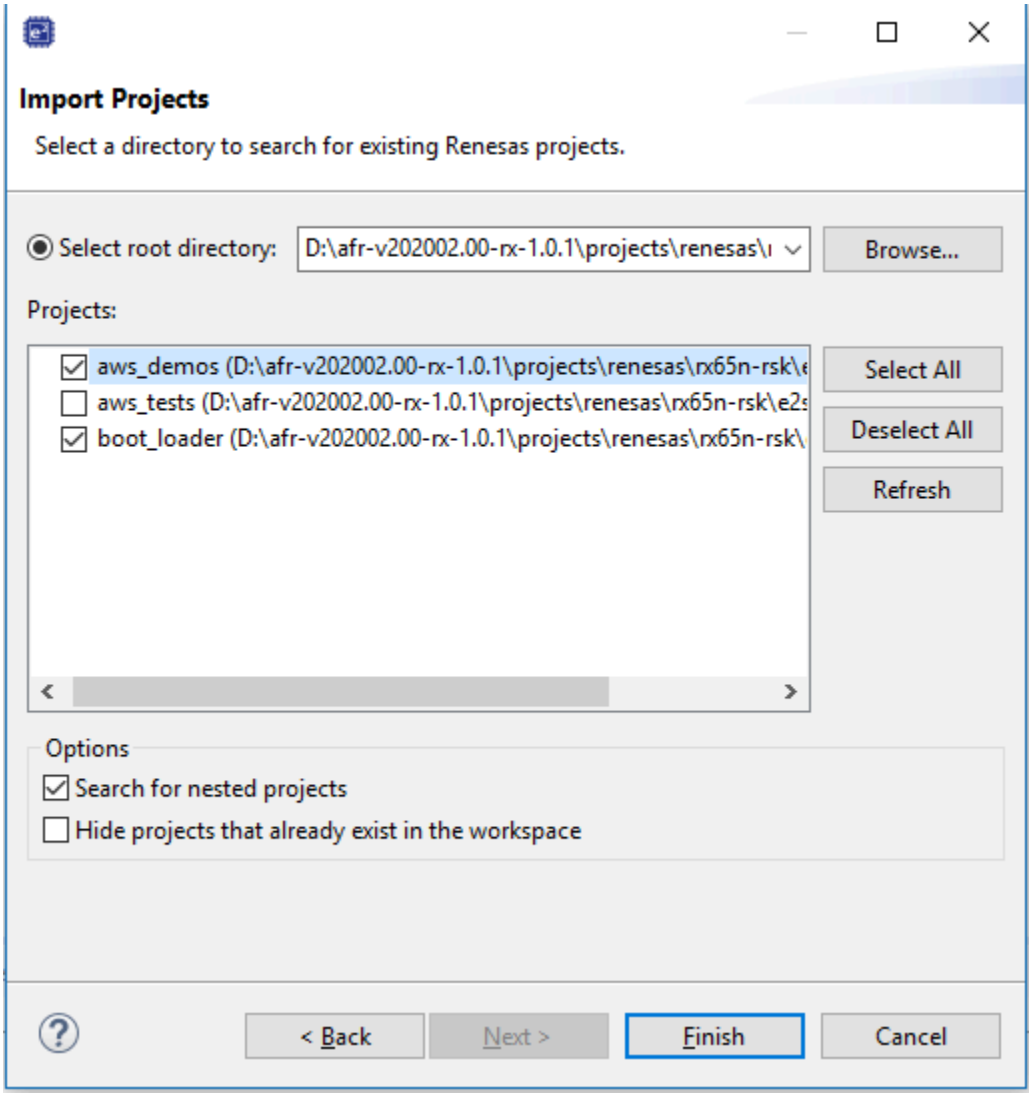
6. Elija Acepto para aceptar el acuerdo de licencia de usuario final.



7. Espere a que finalice la descarga.



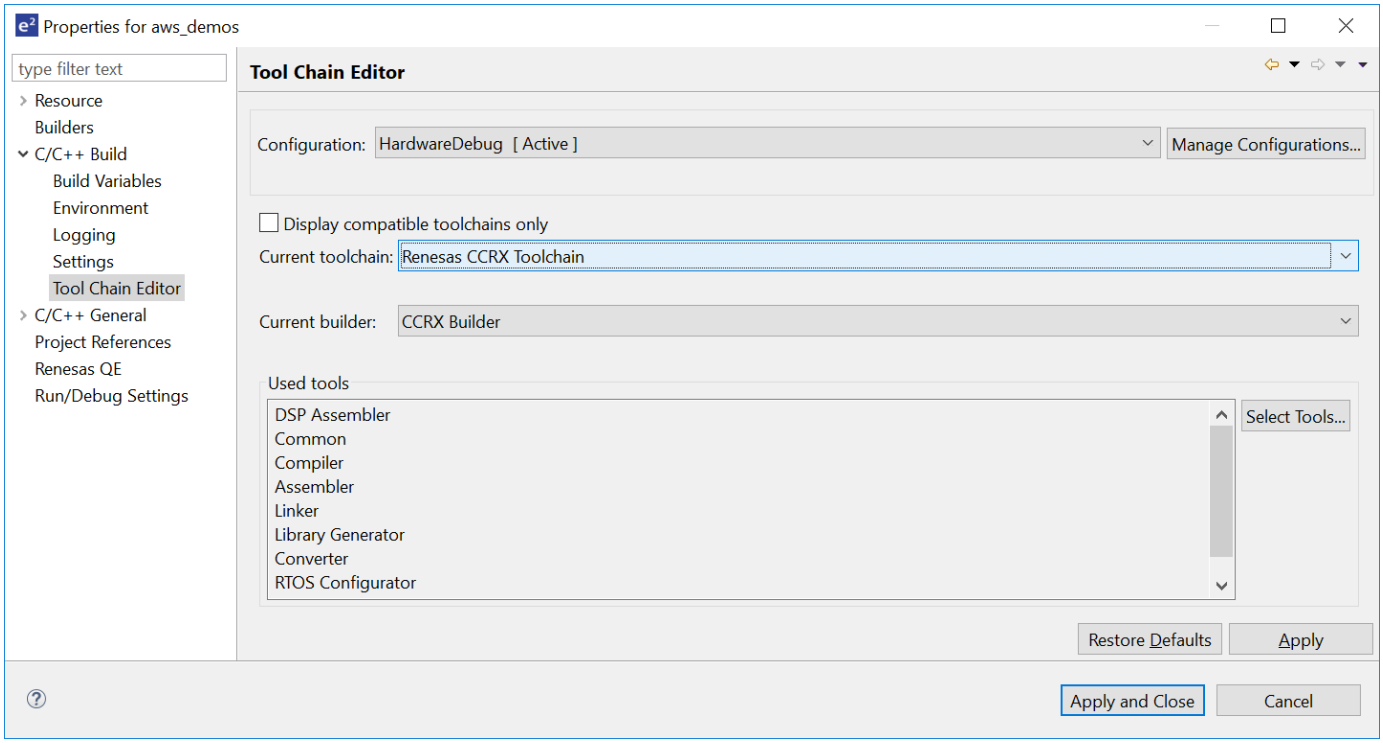
8. Seleccione los proyectos aws_demos y boot_loader y, a continuación, seleccione Finalizar para importarlos.



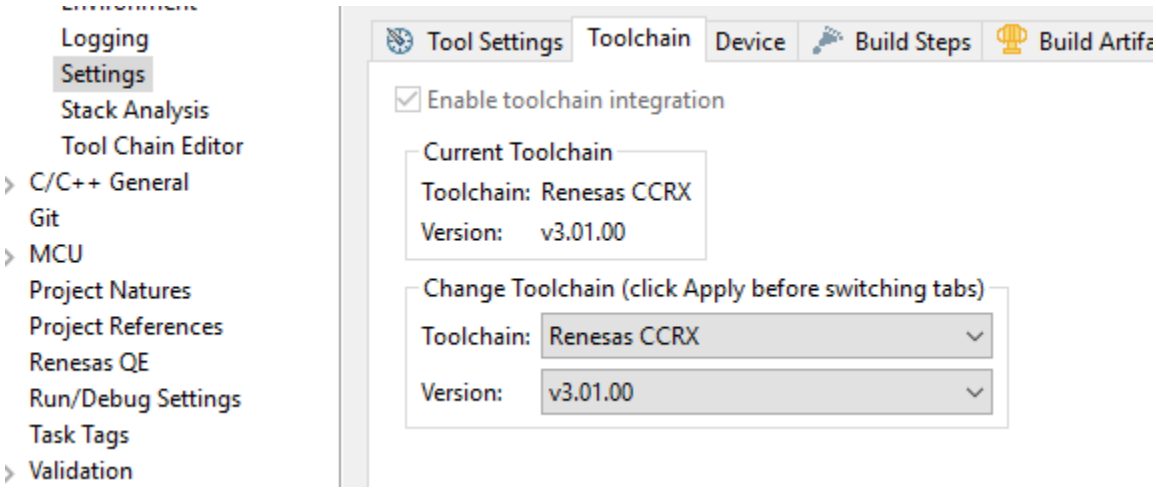
9. Para ambos proyectos, abra las propiedades del proyecto. En el panel de navegación, elija Editor de cadena de herramientas.

    a. Elija la Cadena de herramientas actual.

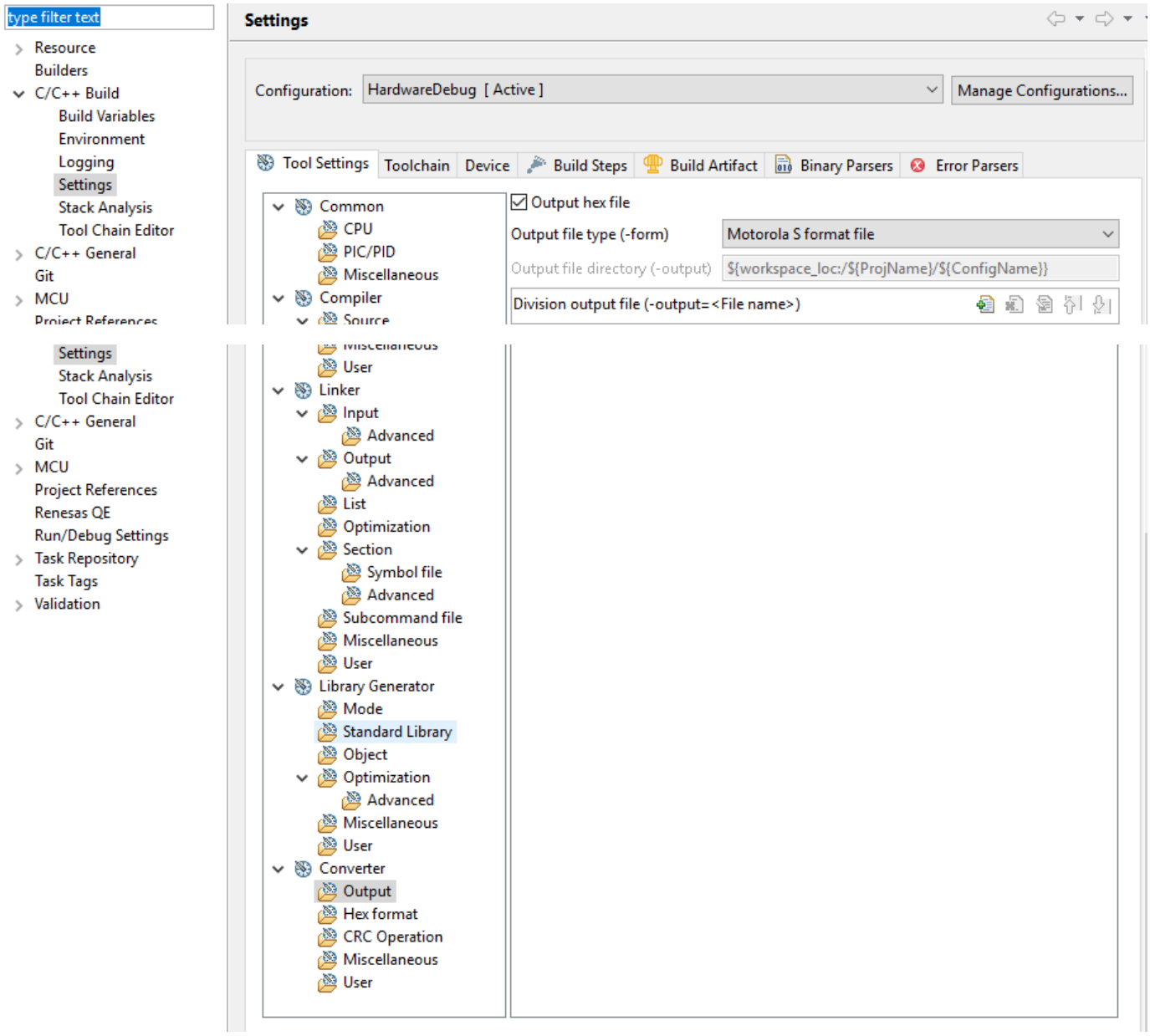
    b. Elija el Creador actual.



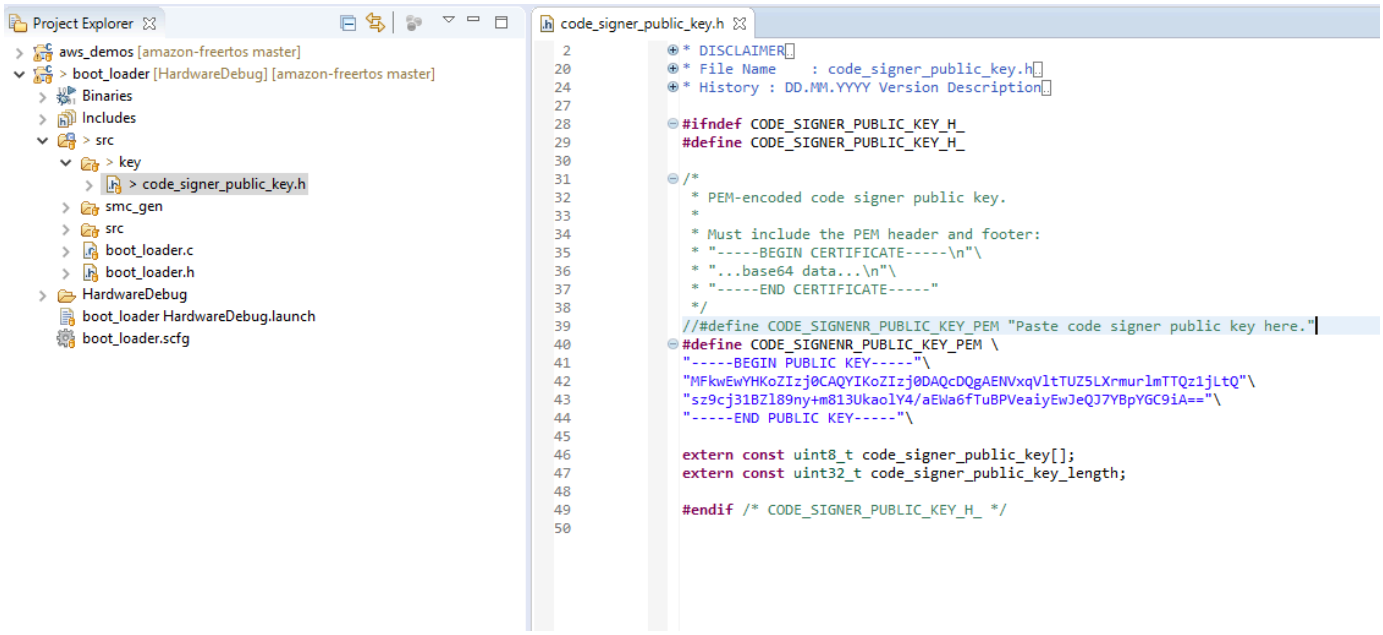
10. En el panel de navegación, seleccione Settings (Configuración). Seleccione la pestaña Cadena de herramientas y, a continuación, elija la Versión de la cadena de herramientas.



Seleccione la pestaña Configuración de la herramienta, expanda Convertidor y, a continuación, seleccione Salida. En la ventana principal, asegúrese de seleccionar Archivo hexadecimal de salida y, a continuación, elija el Tipo de archivo de salida.



11. En el proyecto del cargador de arranque, abra `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h` e introduzca la clave pública. Para obtener información sobre cómo crear una clave pública, consulte Cómo implementar FreeRTOS OTA mediante Amazon Web Services en RX65N y la sección 7.3 “Generación de pares de claves ECDSA-SHA256 con OpenSSL” en la Política de diseño de actualizaciones de firmware de MCU de Renesas.



Para crear el proyecto para crear `boot_loader.mot`.

12. Abra el proyecto `aws_demos`.

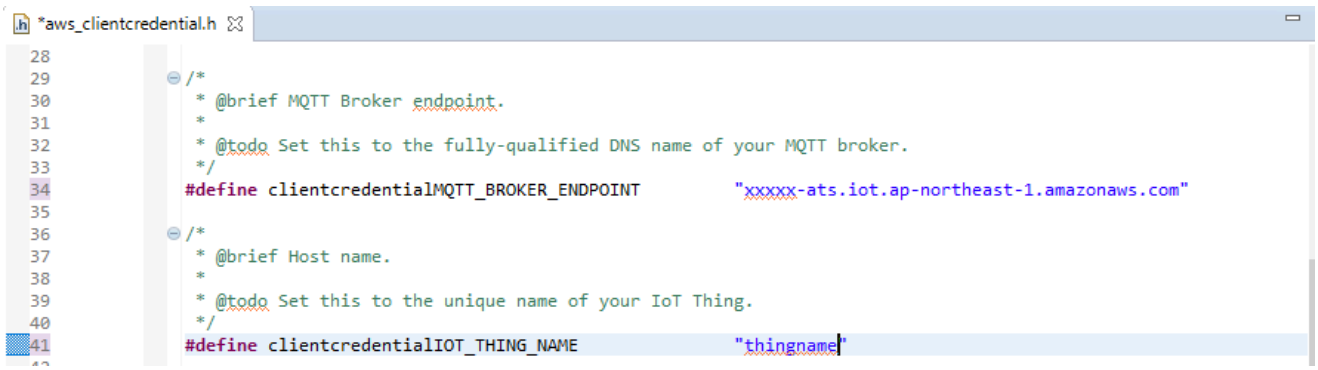
    a. Abra la consola de AWS IoT.

    b. En el panel de navegación izquierdo, elija Configuración. Anote su punto de conexión personalizado en el cuadro de texto Punto de enlace de datos de dispositivo.

    c. Seleccione Administrar y Objetos. Anote el nombre de objeto de AWS IoT de su placa.

    d. En el proyecto `aws_demos`, abra `demos/include/aws_clientcredential.h` y especifique los siguientes valores.

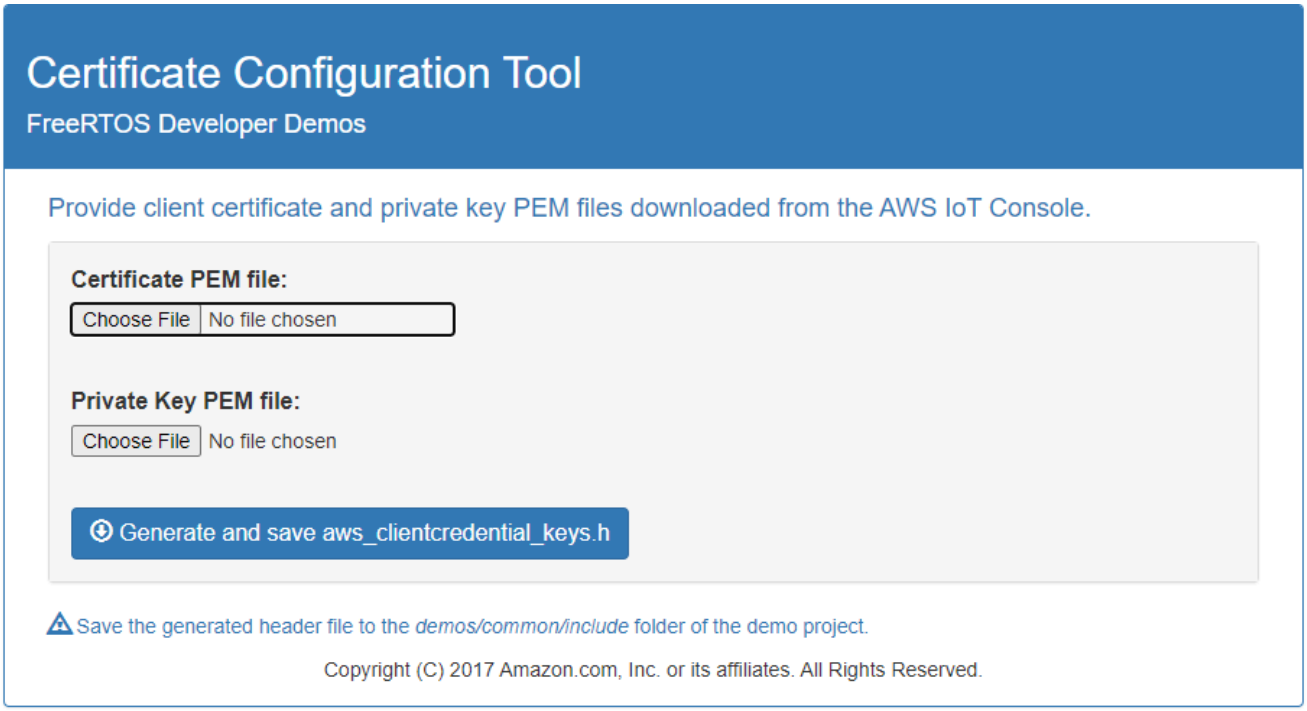
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



e. Abra el archivo `tools/certificate_configuration/CertificateConfigurator.html`.

f. Importe el archivo PEM del certificado y el archivo PEM de clave privada que descargó anteriormente.

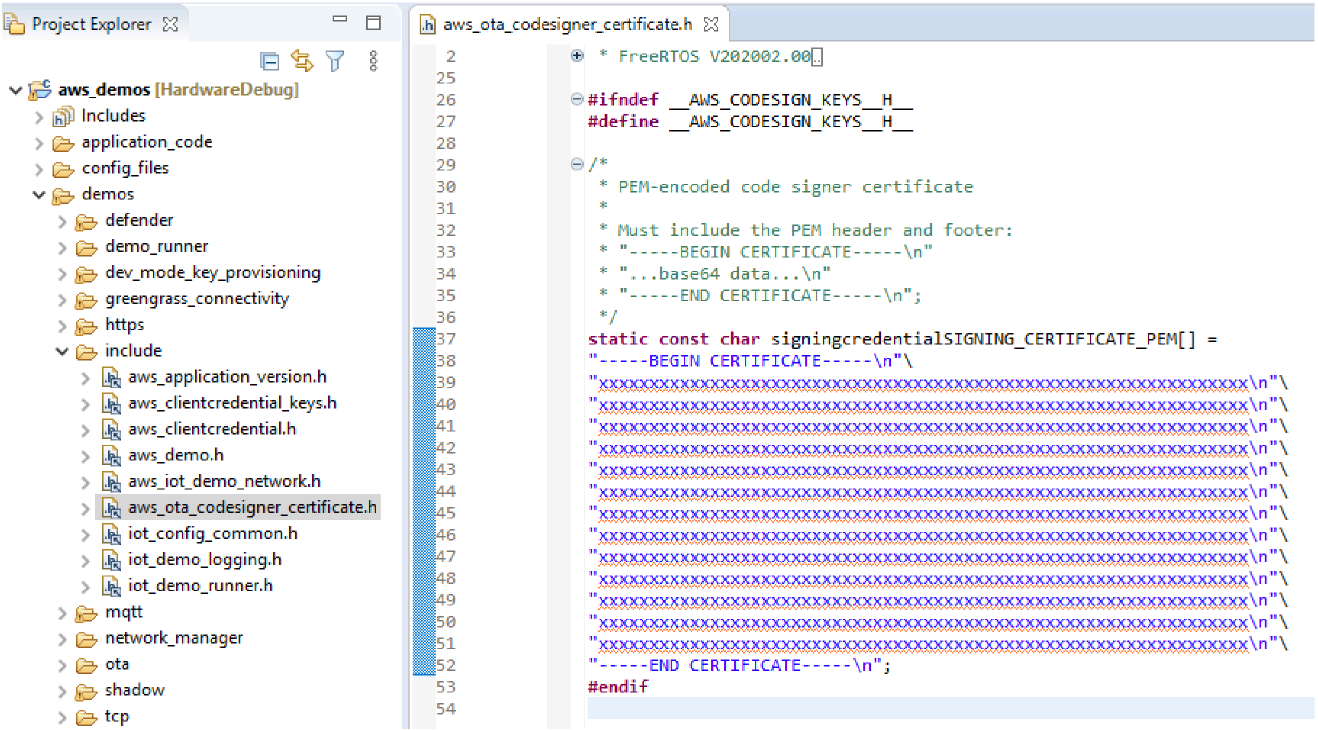
g. Elija Generar y guardar aws_clientcredential_keys.h y guarde el archivo en el directorio `demos/include/`.



h. Abra el archivo `vendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h` y especifique estos valores.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

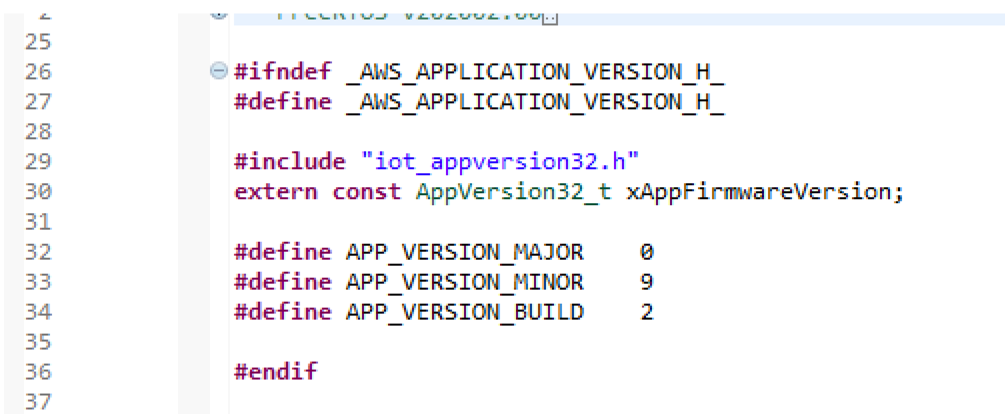
Donde *`your-certificate-key`* es el valor del archivo `secp256r1.crt`. Recuerde añadir "\" después de cada línea de la certificación. Para obtener información sobre cómo crear el archivo `secp256r1.crt`, consulte Cómo implementar FreeRTOS OTA mediante Amazon Web Services en RX65N y la sección 7.3 "Generación de pares de claves ECDSA-SHA256 con OpenSSL" en la Política de diseño de actualizaciones de firmware de MCU de Renesas.



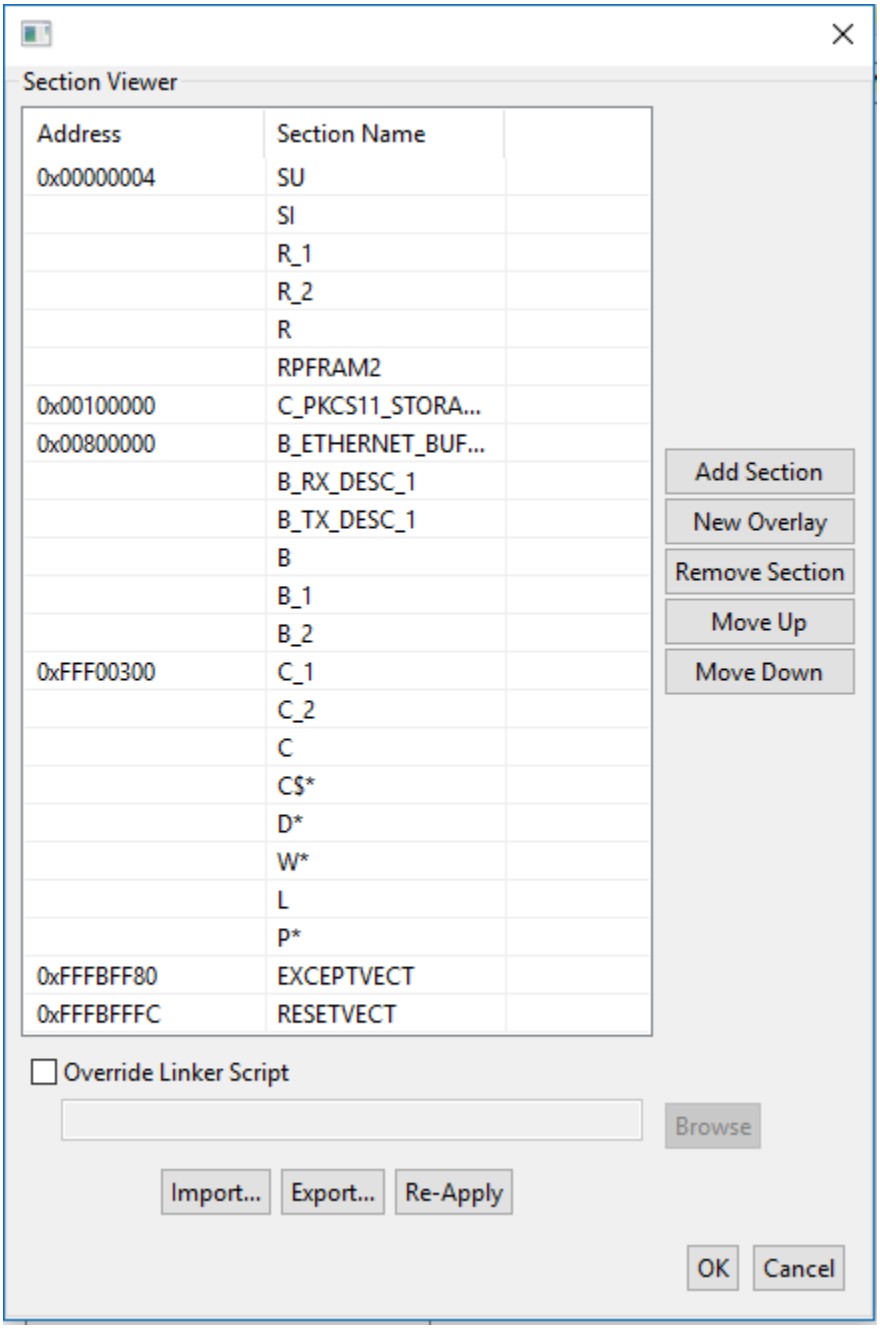
13. Tarea A: Instalar la versión inicial del firmware

   a. Abra el archivo `vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` y defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

   b. Abra el archivo `demos/include/ aws_application_version.h` y configure la versión inicial del firmware en `0.9.2`.



   c. Cambie los siguientes ajustes en Visor de sección.



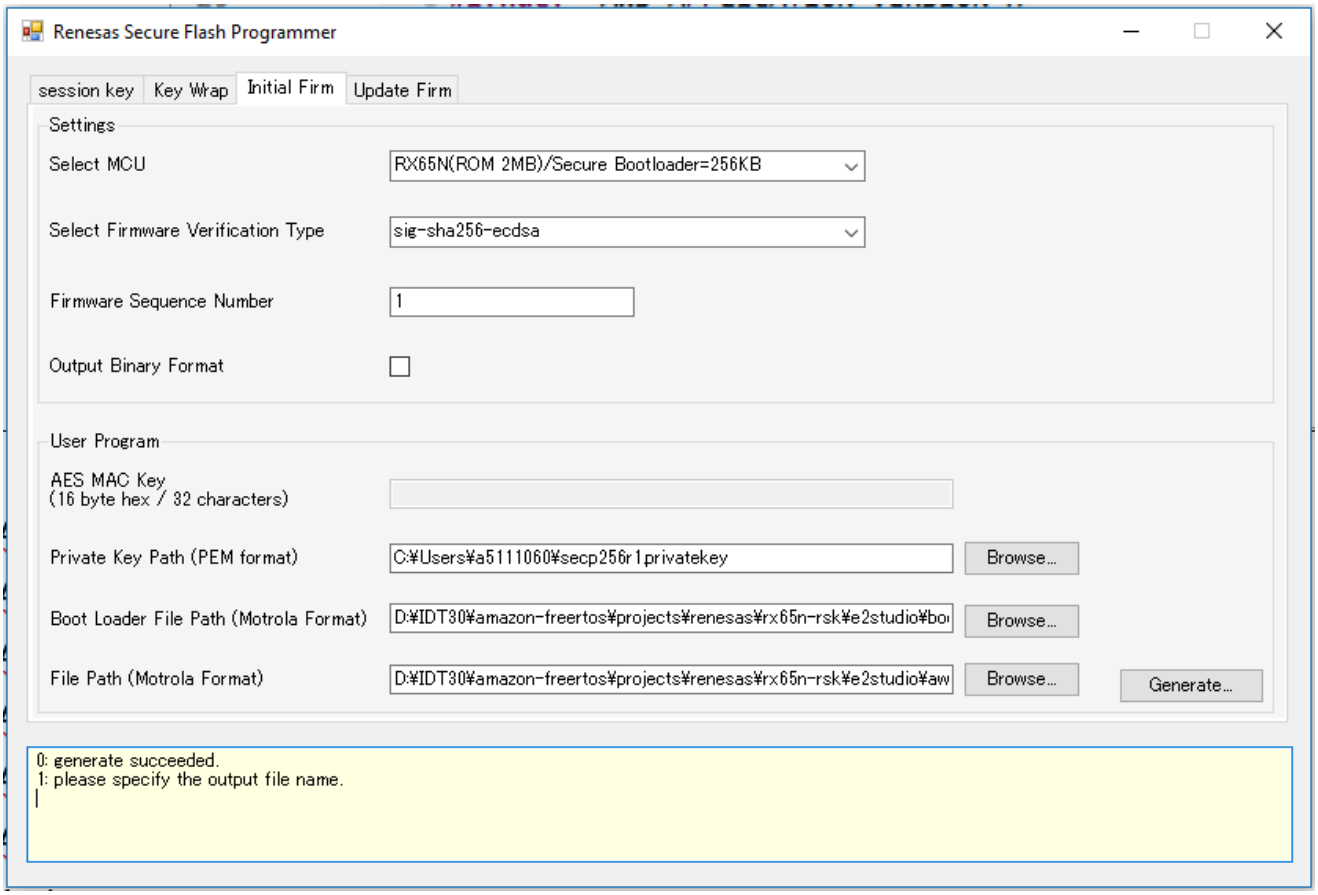
d. Seleccione Crear para crear el archivo `aws_demos.mot`.

14. Cree el archivo `userprog.mot` con el programador Secure Flash de Renesas. `userprog.mot` es una combinación de `aws_demos.mot` y `boot_loader.mot`. Puede instalar este archivo en RX65N-RSK para instalar el firmware inicial.

    a. Descargue https://github.com/renesas/Amazon-FreeRTOS-Tools y abra `Renesas Secure Flash Programmer.exe`.

    b. Seleccione la pestaña Firma inicial y, a continuación, defina los siguientes parámetros:

- Ruta de clave privada - La ubicación de `secp256r1.privatekey`.
- Ruta del archivo del cargador de arranque - La ubicación de `boot_loader.mot` (`projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug`).
- Ruta del archivo - La ubicación de `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).



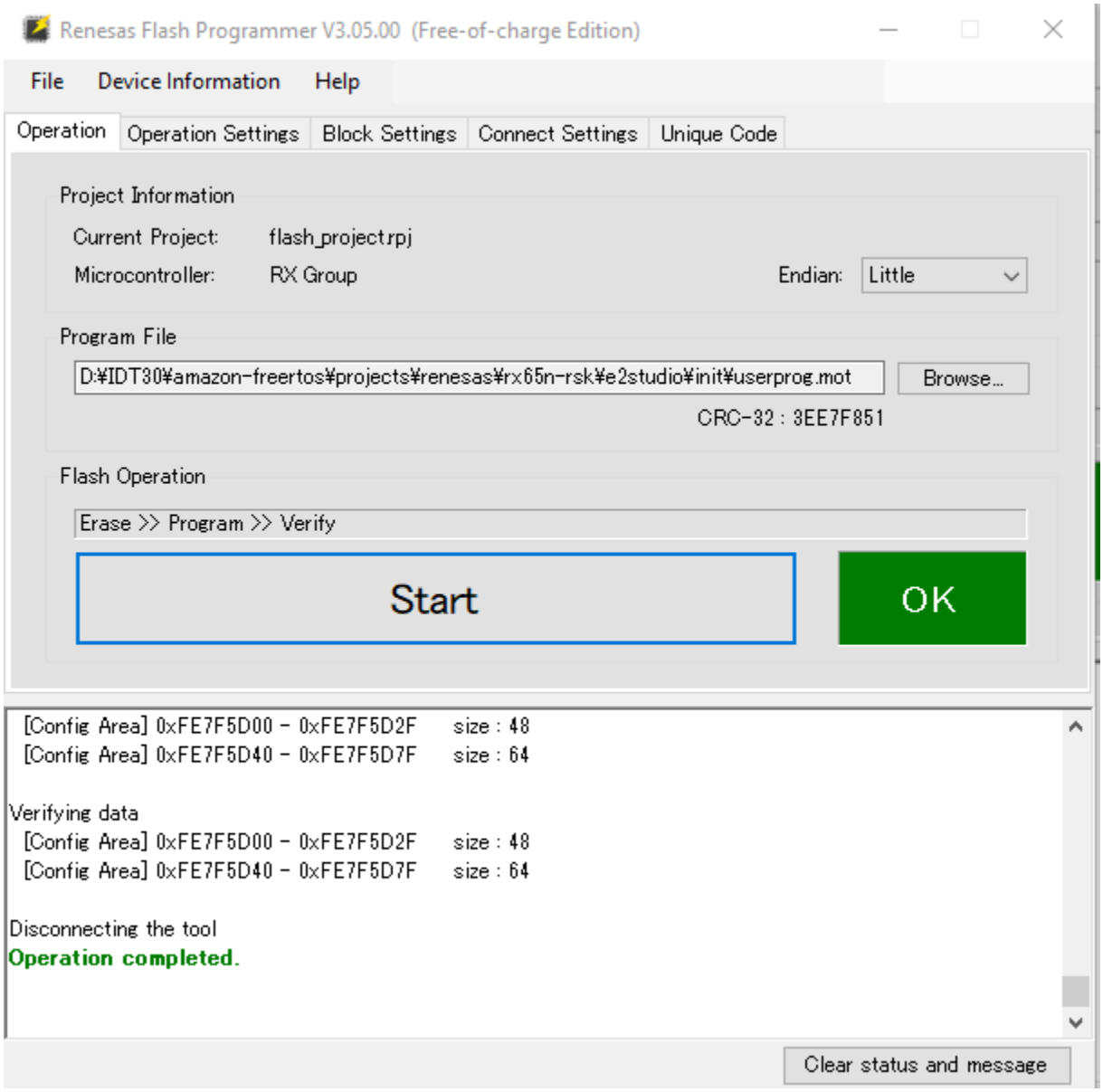
c. Cree un directorio denominado `init_firmware`, genere `userprog.mot` y guárdelo en el directorio `init_firmware`. Compruebe que se haya generado correctamente.

15. Instale el firmware inicial en RX65N-RSK.

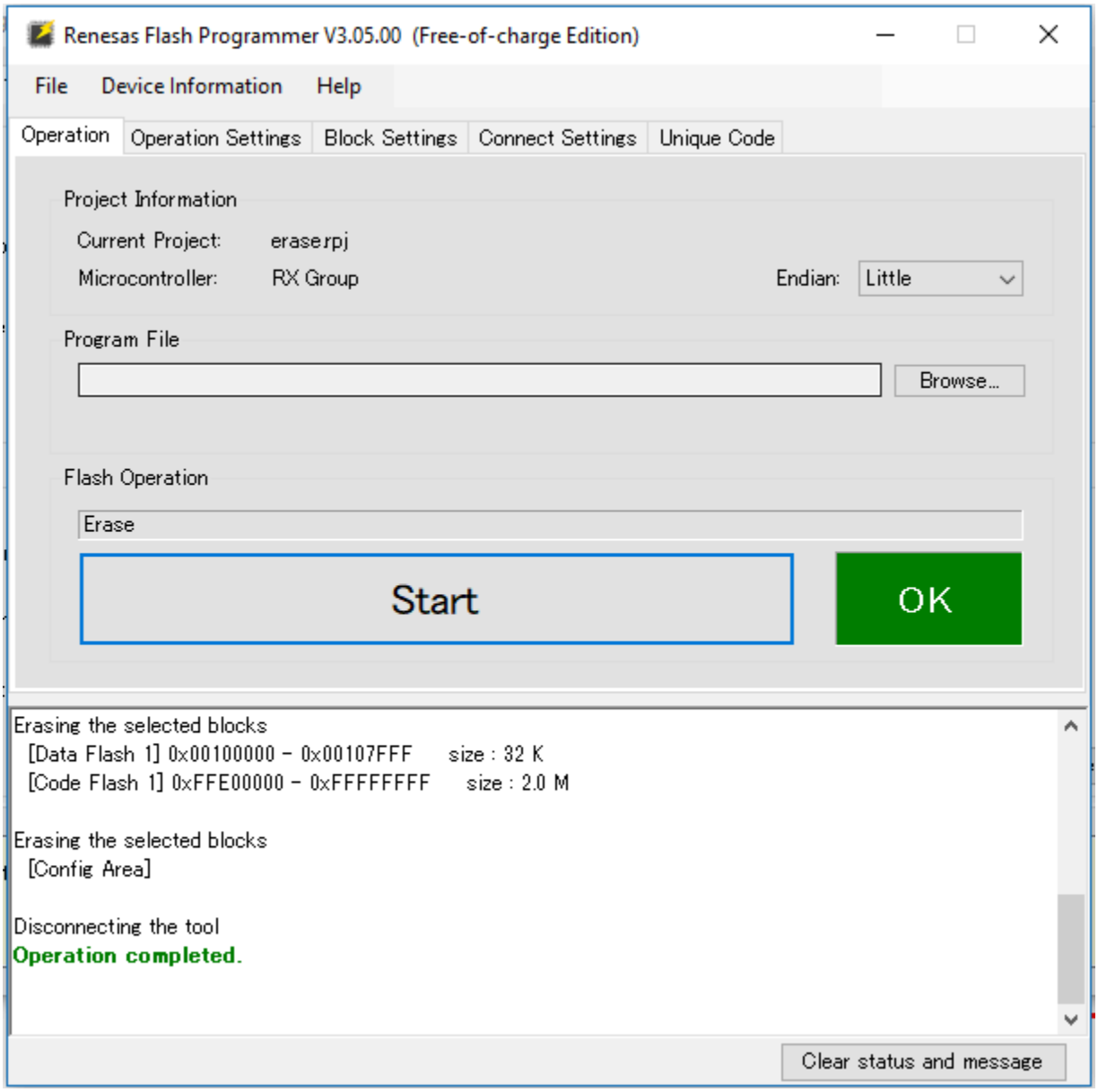
a. Descargue la versión más reciente del programador Renesas Flash (GUI de programación) desde https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html.

b. Abra el archivo `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` para borrar los datos del banco.

c. Seleccione Empezar para borrar el banco.



d. Para instalar `userprog.mot`, seleccione Buscar..., acceda al directorio `init_firmware`, seleccione el archivo `userprog.mot` y elija Iniciar.



16. La versión 0.9.2 (versión inicial) del firmware se instaló en su RX65N-RSK. La placa RX65N-RSK ahora escucha las actualizaciones OTA. Si ha abierto Tera Term en su PC, verá algo parecido a lo siguiente cuando se ejecute el firmware inicial.

```
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xfc [LIFECYCLE_STATE_INSTALLING]
bank info = 1. (start bank = 0)
start installing user program.
```

```
copy secure boot (part1) from bank0 to bank1...OK
copy secure boot (part2) from bank0 to bank1...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_INSTALLING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...
-----------------------------------------------
RX65N secure boot program
-----------------------------------------------
Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
bank info = 0. (start bank = 1)
integrity check scheme = sig-sha256-ecdsa
bank0(execute area) on code flash integrity check...OK
jump to user program
#0 1 [ETHER_RECEI] Deferred Interrupt Handler Task started
1 1 [ETHER_RECEI] Network buffers: 3 lowest 3
2 1 [ETHER_RECEI] Heap: current 234192 lowest 234192
3 1 [ETHER_RECEI] Queue space: lowest 8
4 1 [IP-task] InitializeNetwork returns OK
5 1 [IP-task] xNetworkInterfaceInitialise returns 0
6 101 [ETHER_RECEI] Heap: current 234592 lowest 233392
7 2102 [ETHER_RECEI] prvEMACHandlerTask: PHY LS now 1
8 3001 [IP-task] xNetworkInterfaceInitialise returns 1
9 3092 [ETHER_RECEI] Network buffers: 2 lowest 2
10 3092 [ETHER_RECEI] Queue space: lowest 7
11 3092 [ETHER_RECEI] Heap: current 233320 lowest 233320
12 3193 [ETHER_RECEI] Heap: current 233816 lowest 233120
13 3593 [IP-task] vDHCPProcess: offer c0a80a09ip
14 3597 [ETHER_RECEI] Heap: current 233200 lowest 233000
15 3597 [IP-task] vDHCPProcess: offer c0a80a09ip
16 3597 [IP-task] IP Address: 192.168.10.9
17 3597 [IP-task] Subnet Mask: 255.255.255.0
18 3597 [IP-task] Gateway Address: 192.168.10.1
19 3597 [IP-task] DNS Server Address: 192.168.10.1
20 3600 [Tmr Svc] The network is up and running
21 3622 [Tmr Svc] Write certificate...
22 3697 [ETHER_RECEI] Heap: current 232320 lowest 230904
23 4497 [ETHER_RECEI] Heap: current 226344 lowest 225944
24 5317 [iot_thread] [INFO ][DEMO][5317] ---------STARTING DEMO---------

25 5317 [iot_thread] [INFO ][INIT][5317] SDK successfully initialized.
```

```
26 5317 [iot_thread] [INFO ][DEMO][5317] Successfully initialized the demo. Network
 type for the demo: 4
27 5317 [iot_thread] [INFO ][MQTT][5317] MQTT library successfully initialized.
28 5317 [iot_thread] [INFO ][DEMO][5317] OTA demo version 0.9.2

29 5317 [iot_thread] [INFO ][DEMO][5317] Connecting to broker...

30 5317 [iot_thread] [INFO ][DEMO][5317] MQTT demo client identifier is rx65n-gr-
rose (length 13).
31 5325 [ETHER_RECEI] Heap: current 206944 lowest 206504
32 5325 [ETHER_RECEI] Heap: current 206440 lowest 206440
33 5325 [ETHER_RECEI] Heap: current 206240 lowest 206240
38 5334 [ETHER_RECEI] Heap: current 190288 lowest 190288
39 5334 [ETHER_RECEI] Heap: current 190088 lowest 190088
40 5361 [ETHER_RECEI] Heap: current 158512 lowest 158168
41 5363 [ETHER_RECEI] Heap: current 158032 lowest 158032
42 5364 [ETHER_RECEI] Network buffers: 1 lowest 1
43 5364 [ETHER_RECEI] Heap: current 156856 lowest 156856
44 5364 [ETHER_RECEI] Heap: current 156656 lowest 156656
46 5374 [ETHER_RECEI] Heap: current 153016 lowest 152040
47 5492 [ETHER_RECEI] Heap: current 141464 lowest 139016
48 5751 [ETHER_RECEI] Heap: current 140160 lowest 138680
49 5917 [ETHER_RECEI] Heap: current 138280 lowest 138168
59 7361 [iot_thread] [INFO ][MQTT][7361] Establishing new MQTT connection.
62 7428 [iot_thread] [INFO ][MQTT][7428] (MQTT connection 81cfc8, CONNECT operation
 81d0e8) Wait complete with result SUCCESS.
63 7428 [iot_thread] [INFO ][MQTT][7428] New MQTT connection 4e8c established.
64 7430 [iot_thread] [OTA_AgentInit_internal] OTA Task is Ready.
65 7430 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State [Ready] Event
 [Start] New state [RequestingJob]
66 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8) SUBSCRIBE
 operation scheduled.
67 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Waiting for operation completion.
68 7436 [ETHER_RECEI] Heap: current 128248 lowest 127992
69 7480 [OTA Agent T] [INFO ][MQTT][7480] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Wait complete with result SUCCESS.
70 7480 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/$next/get/accepted
71 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cfc8) SUBSCRIBE
 operation scheduled.
72 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Waiting for operation completion.
```

```
73 7530 [OTA Agent T] [INFO ][MQTT][7530] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Wait complete with result SUCCESS.
74 7530 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/notify-next
75 7530 [OTA Agent T] [prvRequestJob_Mqtt] Request #0
76 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cfc8) MQTT PUBLISH
 operation queued.
77 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cfc8, PUBLISH
 operation 818b80) Waiting for operation completion.
78 7552 [OTA Agent T] [INFO ][MQTT][7552] (MQTT connection 81cfc8, PUBLISH
 operation 818b80) Wait complete with result SUCCESS.
79 7552 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State
 [RequestingJob] Event [RequestJobDocument] New state [WaitingForJob]
80 7552 [OTA Agent T] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:rx65n-gr-rose ]
81 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: execution
82 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobId
83 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobDocument
84 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: afr_ota
85 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: protocols
86 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: files
87 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: filepath
99 7651 [ETHER_RECEI] Heap: current 129720 lowest 127304
100 8430 [iot_thread] [INFO ][DEMO][8430] State: Ready  Received: 1   Queued: 0
 Processed: 0   Dropped: 0
101 9430 [iot_thread] [INFO ][DEMO][9430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
102 10430 [iot_thread] [INFO ][DEMO][10430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
103 11430 [iot_thread] [INFO ][DEMO][11430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
104 12430 [iot_thread] [INFO ][DEMO][12430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
105 13430 [iot_thread] [INFO ][DEMO][13430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
106 14430 [iot_thread] [INFO ][DEMO][14430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
107 15430 [iot_thread] [INFO ][DEMO][15430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
```

17. Tarea B: actualizar la versión del firmware

    a. Abra el archivo demos/include/aws_application_version.h y aumente el valor del token APP_VERSION_BUILD a 0.9.3.

b. Vuelva a compilar el proyecto.

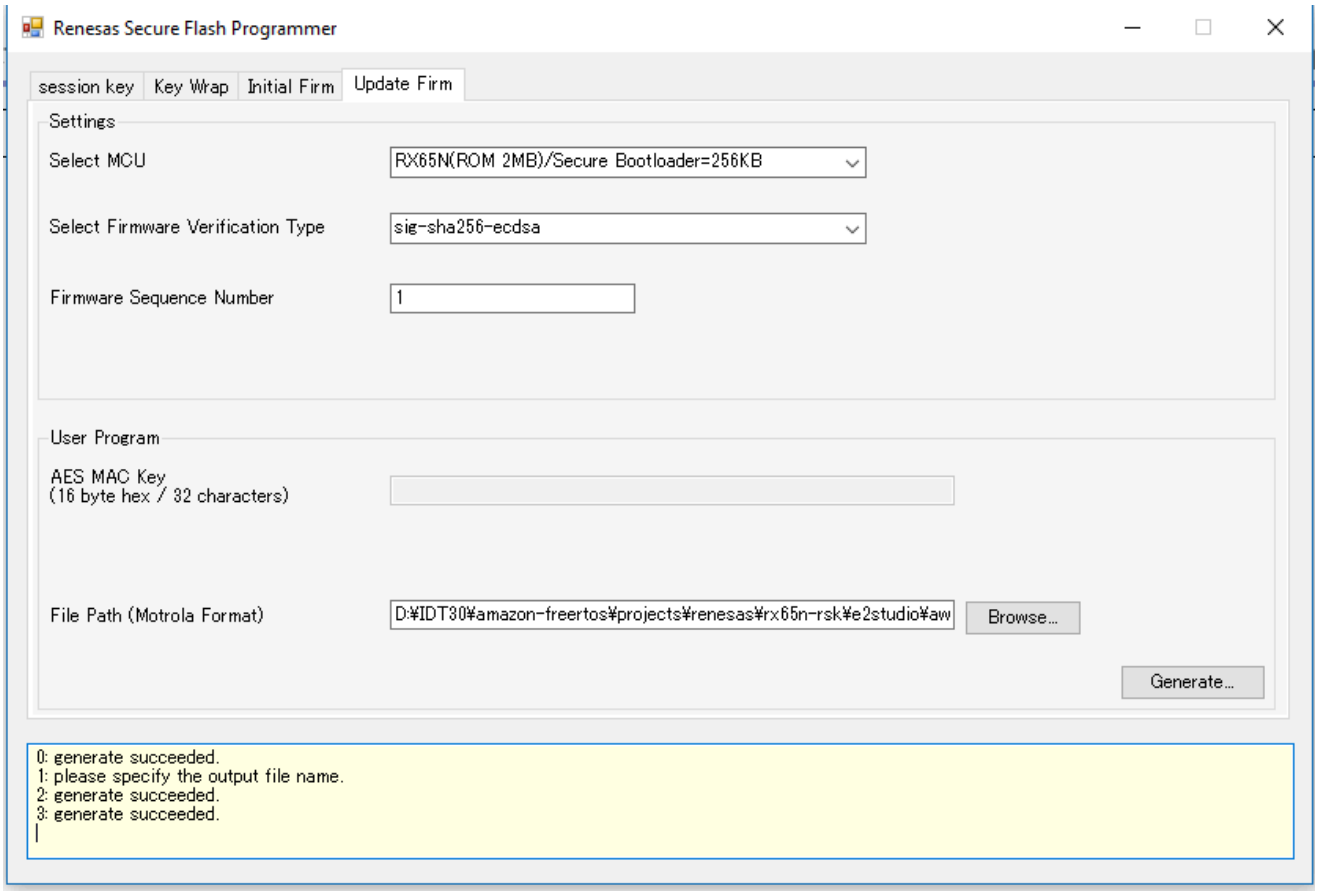
18. Cree el archivo `userprog.rsu` con el programador Secure Flash de Renesas para actualizar la versión de su firmware.

    a. Abra el archivo `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe`.

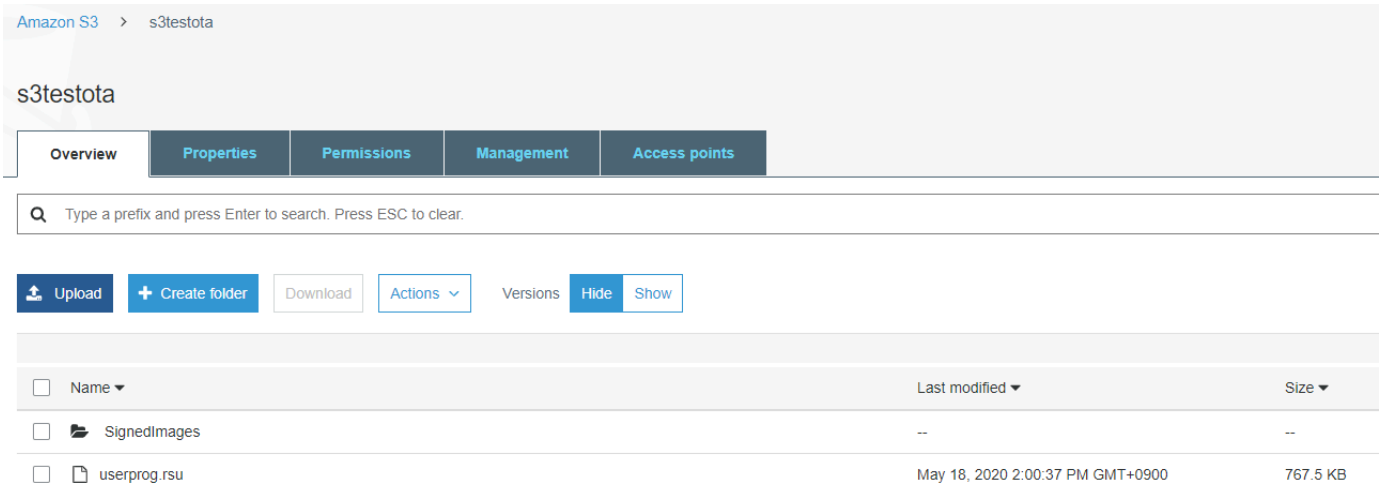
    b. Seleccione la pestaña Actualice el firmware y, a continuación, defina los siguientes parámetros:

    - Ruta del archivo - La ubicación del archivo `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).

    c. Cree un directorio llamado `update _firmware`. Genere `userprog.rsu` y guárdela en el directorio `update_firmware`. Compruebe que se haya generado correctamente.



19. Cargue la actualización del firmware, `userproj.rsu`, en un bucket de Amazon S3 tal y como se describe enCreación de un bucket de Amazon S3 para almacenar la actualización.



20. Cree un trabajo para actualizar el firmware del RX65N-RSK.

    Trabajos de AWS IoT es un servicio que notifica a uno o más dispositivos conectados un trabajo pendiente. Puede usar un trabajo para administrar una flota de dispositivos, actualizar el firmware y los certificados de seguridad de sus dispositivos o realizar tareas administrativas, como reiniciar los dispositivos y realizar diagnósticos.

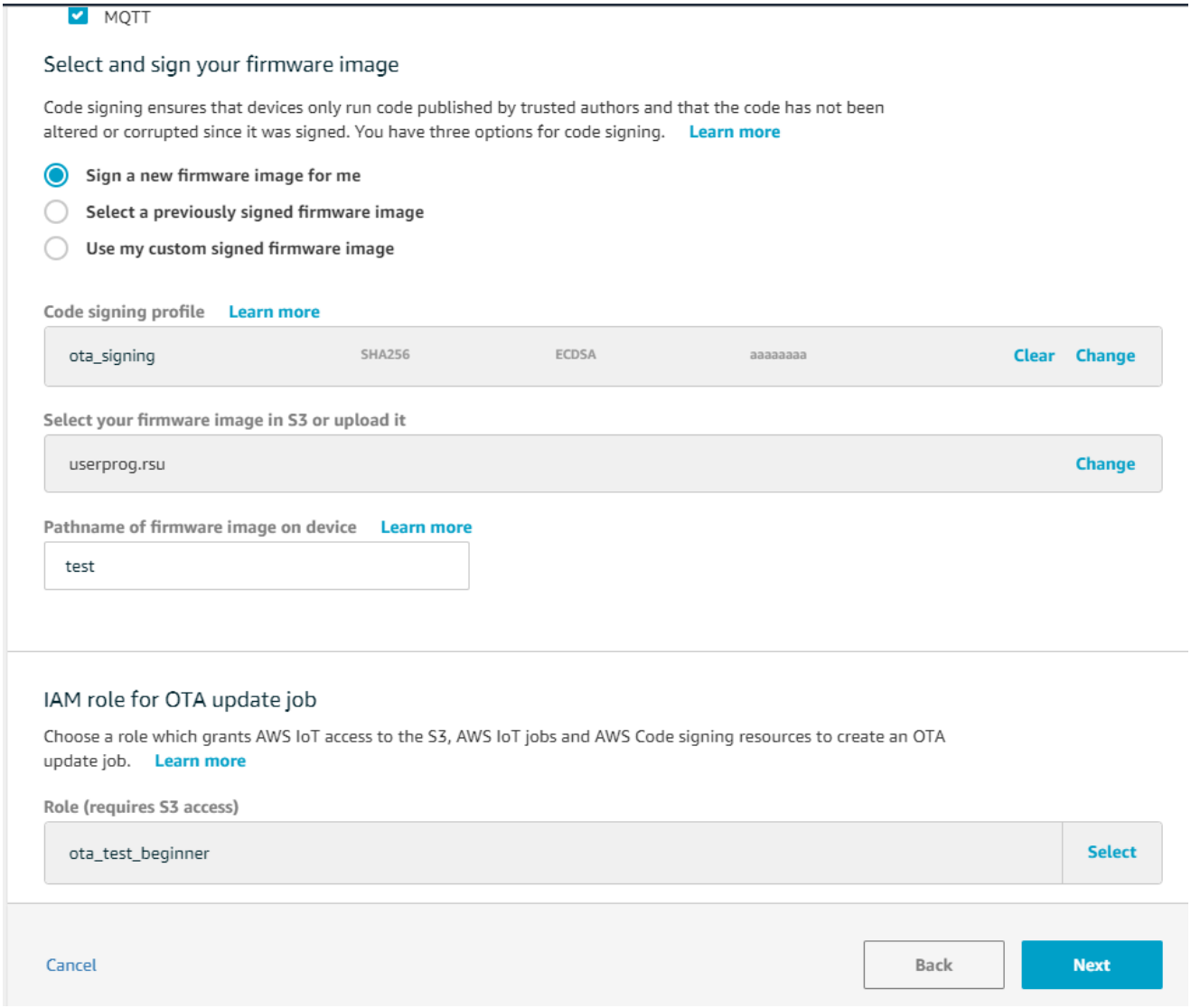
    a. Inicie sesión en la consola de AWS IoT. En el panel de navegación, elija Administrar y, a continuación, Trabajos.

    b. Elija Crear y, a continuación, elija Crear trabajo de actualización OTA. Seleccione un objeto y, a continuación, elija Siguiente.

    c. Cree un trabajo de actualización OTA de FreeRTOS de la siguiente manera:

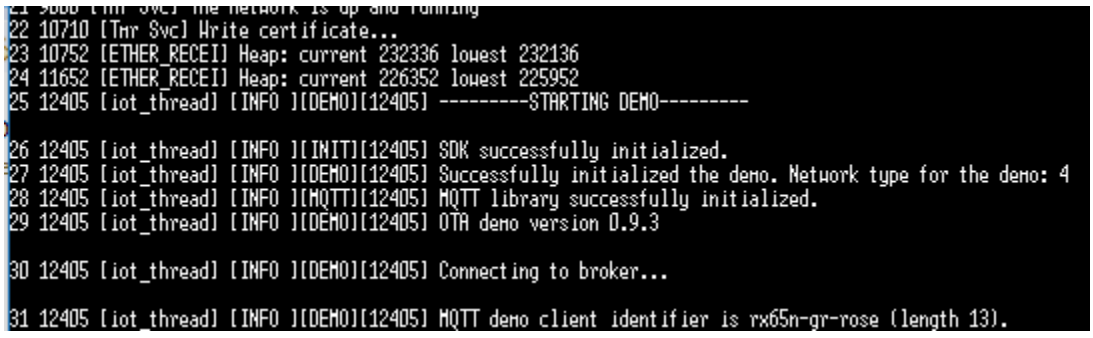
    - Elija MQTT.
    - Seleccione el perfil de firma de código que ha creado en la sección anterior.
    - Seleccione la imagen de firmware que ha cargado en un bucket de Amazon S3.
    - En Nombre de la ruta de la imagen de firmware en el dispositivo, introduzca **`test`**.
    - Elija el rol de IAM que ha creado en la sección anterior.

    d. Elija Siguiente.

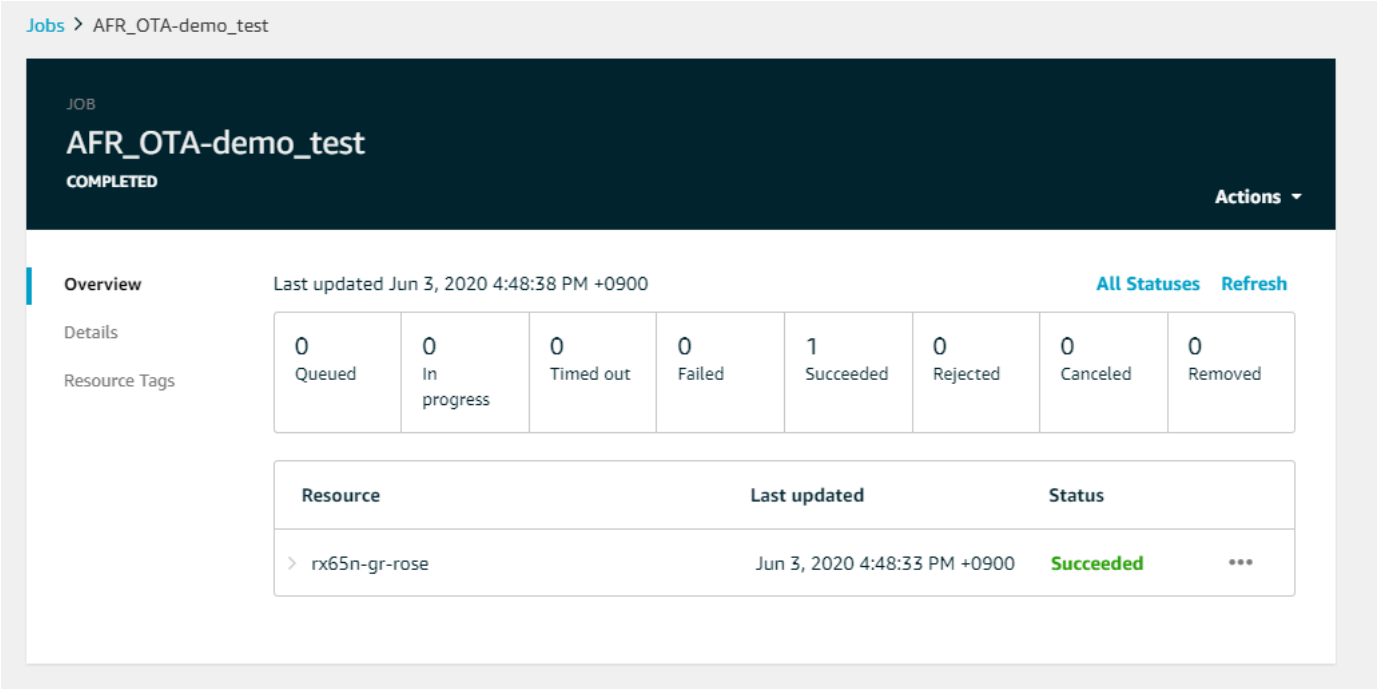


e. Introduzca el ID y, a continuación, elija Crear.

21. Vuelva a abrir Tera Term para comprobar que el firmware se actualizó correctamente a la versión de demostración 0.9.3 de OTA.



22. En la consola AWS IoT, compruebe que el estado del trabajo es Correcto.



Tutorial: Realización de actualizaciones OTA en el Espressif ESP32 utilizando Bluetooth de bajo consumo de FreeRTOS

> **Important**
>
> Esta integración de referencia está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

En este tutorial, se muestra cómo actualizar un microcontrolador Espressif ESP32 que esté conectado a un proxy Bluetooth de bajo consumo de MQTT en un dispositivo Android. Actualiza el dispositivo mediante trabajos de actualización inalámbrica de AWS IoT. El dispositivo se conecta a AWS IoT mediante las credenciales de Amazon Cognito introducidas en la aplicación de demostración de Android. Un operador autorizado inicia la actualización OTA desde la nube. Cuando el dispositivo se conecta a través de la aplicación de demostración de Android, se inicia la actualización OTA y se actualiza el firmware del dispositivo.

Las versiones 2019.06.00 principal y posteriores de FreeRTOS incluyen compatibilidad con el proxy MQTT de Bluetooth de bajo consumo que se puede utilizar para el aprovisionamiento de Wi-Fi y las

conexiones seguras a los servicios de AWS IoT. Al usar la característica Bluetooth de bajo consumo, puede crear dispositivos de bajo consumo que se pueden emparejar a un dispositivo móvil para tener conectividad sin necesidad de Wi-Fi. Los dispositivos pueden comunicarse con MQTT mediante la conexión a través de los SDK de Bluetooth de bajo consumo de Android o iOS que utilizan perfiles de perfil de acceso genérico (GAP) y de atributos genéricos (GATT).

Estos son los pasos que seguiremos para permitir las actualizaciones OTA a través de Bluetooth de bajo consumo:

1. Configurar el almacenamiento: cree un bucket y políticas de Amazon S3 y configure un usuario que pueda realizar actualizaciones.
2. Crear un certificado de firma de código: cree un certificado de firma y permita al usuario firmar las actualizaciones del firmware.
3. Configurar la autenticación de Amazon Cognito: cree un proveedor de credenciales, un grupo de usuarios y un acceso de aplicaciones al grupo de usuarios.
4. Configurar Freertos: configure Bluetooth de bajo consumo, las credenciales del cliente y el certificado público de firma de código.
5. Configurar una aplicación para Android: configure el proveedor de credenciales, el grupo de usuarios e implemente la aplicación en un dispositivo Android.
6. Ejecutar el script de actualización OTA: para iniciar una actualización OTA, utilice el script de actualización OTA.

Para obtener más información sobre cómo funcionan las actualizaciones, consulte Actualizaciones vía inalámbrica de FreeRTOS. Para obtener información adicional sobre cómo configurar la funcionalidad de proxy MQTT de Bluetooth de bajo consumo, consulte la publicación Uso de Bluetooth de bajo consumo con FreeRTOS en Espressif ESP32 de Richard Kang.

Requisitos previos

Para realizar los pasos de este tutorial necesitará los siguientes recursos:

- Una placa de desarrollo ESP32.
- Un cable microUSB a USB A.
- Una cuenta de AWS (el nivel gratuito es suficiente).
- Un teléfono Android con Android v 6.0 o posterior y Bluetooth versión 4.2 o posterior.

En tu equipo de desarrollo necesita:

- Espacio en disco suficiente (~500 Mb) para la cadena de herramientas de Xtensa y el código fuente y ejemplos de FreeRTOS.
- Android Studio instalado.
- La AWS CLI instalada.
- Python3 instalado.
- El kit de desarrollo de software (SDK) de AWS boto3 para Python.

En los pasos de este tutorial se supone que la cadena de herramientas de Xtensa, el código ESP-IDF y el código FreeRTOS están instalados en el directorio `/esp` de su directorio principal. Debe añadir `~/esp/xtensa-esp32-elf/bin` a la variable `$PATH`.

Paso 1: Configurar almacenamiento

1. Creación de un bucket de Amazon S3 para almacenar la actualización con el control de versiones activado para almacenar las imágenes del firmware.
2. Crear un rol de servicio de actualizaciones OTA y añada las siguientes políticas administradas al rol:
   - AWSIotLogging
   - AWSIotRuleActions
   - AWSIotThingsRegistration
   - AWSFreeRTOSOTAUpdate
3. Cree un usuario que pueda realizar actualizaciones OTA. Este usuario puede firmar e implementar actualizaciones de firmware en los dispositivos de IoT de la cuenta y tiene acceso para realizar actualizaciones OTA en todos los dispositivos. El acceso debe estar limitado a entidades de confianza.
4. Siga los pasos para Crear una política de usuario de OTA y asóciela a su usuario.

Paso 2: Crear el certificado de firma de código

1. Cree un bucket de Amazon S3 con el control de versiones habilitado para almacenar las imágenes del firmware.

2. Cree un certificado de firma de código que pueda usarse para firmar el firmware. Anote el Nombre de recurso de Amazon (ARN) del certificado cuando se importe.

```
aws acm import-certificate --profile=ota-update-user --certificate file://
ecdsasigner.crt --private-key file://ecdsasigner.key
```

Ejemplo de resultados:

```
{
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"
}
```

Utilizará el ARN más adelante para crear un perfil de firma. Si lo desea, puede crear el perfil ahora con el siguiente comando:

```
aws signer put-signing-profile --profile=ota-update-user --profile-
name esp32Profile --signing-material certificateArn=arn:aws:acm:us-
east-1:account:certificate/certid --platform AmazonFreeRTOS-Default --signing-
parameters certname=/cert.pem
```

Ejemplo de resultados:

```
{
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"
}
```

Paso 3: Configuración de autenticación de Amazon Cognito

Creación de una política de AWS IoT

1. Inicie sesión en la consola de AWS IoT.
2. En la esquina superior derecha de la consola, elija Mi cuenta. En Configuración de la cuenta, anote el ID de 12 dígitos de la cuenta.
3. En el panel de navegación izquierdo, elija Configuración. En Punto de enlace de datos de dispositivo, anote el valor del punto de conexión. El punto de conexión debería ser similar a `xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com`. En este ejemplo, la región de AWS es “us-west-2”.

4. En el panel de navegación izquierdo, elija Seguridad, Políticas y, a continuación, Crear. Si no tiene ninguna política en su cuenta, verá el mensaje “Aún no tiene ninguna política” y podrá elegir Crear una política.
5. Introduzva un nombre para la política, por ejemplo, “esp32_mqtt_proxy_iot_policy”.
6. En la sección Añadir declaraciones, elija Modo avanzado. Copie y pegue la siguiente política JSON en la ventana del editor de políticas. Sustituya `aws-account-id` por su ID de cuenta de `aws-region` y por su región (por ejemplo, “us-west-2”).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    }
  ]
}
```

7. Seleccione Crear.

Creación de un objeto de AWS IoT

1. Inicie sesión en la consola de AWS IoT.

2. En el panel de navegación de la izquierda, elija Manage (Administrar) y, a continuación, Things (Objetos).
3. En la esquina superior derecha, elija Crear. Si no tiene ningún objeto registrado en su cuenta, aparecerá el mensaje “Aún no tiene ningún objeto” y podrá seleccionar Registrar un objeto.
4. En la página Creación de objetos de AWS IoT, elija Crear un solo objeto.
5. En la página Añadir su dispositivo al registro de objetos, escriba un nombre para el objeto (por ejemplo, “esp32-ble”). Se permiten caracteres alfanuméricos, guiones (-) y guiones bajos (_). Elija Siguiente.
6. En la página Añadir un certificado para el objeto, en Omitir certificado y crear objeto, elija Crear un objeto sin certificado. Como utilizamos la aplicación móvil proxy BLE que utiliza una credencial de Amazon Cognito para la autenticación y la autorización, no se requiere ningún certificado de dispositivo.

Creación de un cliente de aplicación de Amazon Cognito

1. Inicie sesión en la consola de Amazon Cognito.
2. En el banner de navegación de la parte superior derecha, seleccione Crear un grupo de usuarios.
3. Introduzca el nombre del grupo (por ejemplo, “esp32_mqtt_proxy_user_pool”).
4. Elija Review defaults.
5. En Clientes de aplicación, elija Agregar cliente de aplicación y, a continuación, elija Agregar un cliente de aplicación.
6. Introduzca un nombre de cliente de aplicación (por ejemplo, “mqtt_app_client”).
7. Asegúrese de que esté seleccionada la opción Generar secreto de cliente.
8. Elija Create app client (Crear cliente de aplicación).
9. Elija Return to pool details (Volver a los detalles del grupo).
10. En la página Revisar, elija Crear grupo. Debería ver un mensaje que indica: “El grupo de usuarios se ha creado correctamente”. Anote el ID del grupo.
11. En el panel de navegación, elija Clientes de aplicación.
12. Elija Mostrar detalles. Anote el ID y el secreto del cliente de aplicación.

Creación de un grupo de identidades en Amazon Cognito

1. Inicie sesión en la consola de Amazon Cognito.
2. Elija Create new identity pool.
3. Introduzca un nombre para el grupo de identidades (por ejemplo, “mqtt_proxy_identity_pool”).
4. Amplíe Proveedores de autenticación.
5. Seleccione la pestaña Cognito.
6. Introduzca el ID del grupo de usuarios y el ID de cliente de la aplicación que anotó en los pasos anteriores.
7. Elija Create Pool (Crear grupo).
8. En la página siguiente, para crear roles nuevos para las identidades autenticadas y no autenticadas, elija Permitir.
9. Anote el ID del grupo de identidades, que tiene el formato `us-east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Asociación de una política de IAM a la identidad autenticada

1. Abra la consola de Amazon Cognito.
2. Seleccione el grupo de identidades que acaba de crear (por ejemplo, “mqtt_proxy_identity_pool”).
3. Elija Edit identity pool (Editar grupo de identidades).
4. Anote el rol de IAM asignado al rol autenticado (por ejemplo, “Cognito_mqtt_proxy_identity_poolAuth_Role”).
5. Abra la consola de IAM.
6. Seleccione Roles (Roles) en el panel de navegación.
7. Busque el rol asignado (por ejemplo, “Cognito_mqtt_proxy_identity_poolAuth_Role”) y selecciónelo.
8. Elija Añadir política en línea y, a continuación, seleccione la pestaña JSON.
9. Escriba la siguiente política:

```
{
       "Version": "2012-10-17",
       "Statement": [
       {
```

```
            "Effect": "Allow",
            "Action": [
               "iot:AttachPolicy",
               "iot:AttachPrincipalPolicy",
               "iot:Connect",
               "iot:Publish",
               "iot:Subscribe"
            ],
            "Resource": "*"
         }]
      }
```

10. Elija Review Policy (Revisar la política).
11. Introduzva un nombre para la política, por ejemplo, “mqttProxyCognitoPolicy”).
12. Elija Create Policy (Crear política).

Paso 4: Configurar Amazon Freertos

1. Descargue la última versión del código de Amazon FreeRTOS del repositorio de GitHub de FreeRTOS.
2. Para activar la demostración de la actualización OTA, siga los pasos que se indican en Cómo empezar con el Espressif ESP32- C y el ESP-WROVER-KIT DevKit.
3. Realice estas modificaciones adicionales en los siguientes archivos:

   a. Abra `vendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h` y defina `CONFIG_OTA_UPDATE_DEMO_ENABLED`.

   b. Abra `vendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h` y cambie `democonfigNETWORK_TYPES` a `AWSIOT_NETWORK_TYPE_BLE`.

   c. Abra `demos/include/aws_clientcredential.h` e introduzca la URL de su punto de conexión para `clientcredentialMQTT_BROKER_ENDPOINT`.

      Introduzca el nombre del objeto para `clientcredentialIOT_THING_NAME` (por ejemplo, “esp32-ble”). No es necesario añadir certificados cuando se utilizan las credenciales de Amazon Cognito.

   d. Abra `vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h` y cambie `configSUPPORTED_NETWORKS` y `configENABLED_NETWORKS` para que solo incluya `AWSIOT_NETWORK_TYPE_BLE`.

e. Abra el archivo `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` e introduzca su certificado.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

La aplicación debería iniciarse e imprimir la versión de demostración:

```
11 13498 [iot_thread] [INFO ][DEMO][134980] Successfully initialized the demo.
 Network type for the demo: 2
12 13498 [iot_thread] [INFO ][MQTT][134980] MQTT library successfully initialized.
13 13498 [iot_thread] OTA demo version 0.9.20
14 13498 [iot_thread] Creating MQTT Client...
```

Paso 5: Configurar una aplicación para Android

1. Descargue el SDK de Bluetooth de bajo consumo para Android y una aplicación de muestra del repositorio de GitHub amazon-freertos-ble-android-sdk.
2. Abra el archivo `app/src/main/res/raw/awsconfiguration.json` y rellene el ID del grupo, la región, el AppClientID y el AppClientSecret siguiendo las instrucciones del siguiente ejemplo de JSON.

```
{
  "UserAgent": "MobileHub/1.0",
  "Version": "1.0",
  "CredentialsProvider": {
    "CognitoIdentity": {
      "Default": {
        "PoolId": "Cognito->Manage Identity Pools->Federated Identities-
>mqtt_proxy_identity_pool->Edit Identity Pool->Identity Pool ID",
        "Region": "Your region (for example us-east-1)"
      }
    }
  },

  "IdentityManager": {
    "Default": {}
  },

  "CognitoUserPool": {
```

```
    "Default": {
      "PoolId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> PoolId",
      "AppClientId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> App clients ->Show Details",
      "AppClientSecret": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool
 -> General Settings -> App clients ->Show Details",
      "Region": "Your region (for example us-east-1)"
    }
  }
}
```

3. Abra `app/src/main/java/software/amazon/freertos/DemoConstants.java` e introduzca el nombre de la política que creó anteriormente (por ejemplo, *esp32_mqtt_proxy_iot_policy*) y también la región (por ejemplo, *us-east-1*).
4. Cree e instale la aplicación de demostración.
   a. En Android Studio, seleccione Crear y, luego, Crear aplicación de módulo.
   b. Elija Ejecutar y, a continuación, Ejecutar aplicación. Puede ir al panel de la ventana de logcat en Android Studio para monitorear los mensajes de registro.
   c. En el dispositivo Android, crea una cuenta desde la pantalla de inicio de sesión.
   d. Crear un usuario. Si ya existe un usuario, introduzca las credenciales.
   e. Permita que la demostración de Amazon FreeRTOS acceda a la ubicación del dispositivo.
   f. Busque dispositivos Bluetooth de bajo consumo.
   g. Mueva el control deslizante del dispositivo que se encuentra a Activado.
   h. Pulse y en la consola de depuración del puerto serie para ESP32.
   i. Elija Emparejar y conectar.
5. El enlace Más… se activa una vez establecida la conexión. El estado de la conexión debería cambiar a “BLE_CONNECTED” en el logcat del dispositivo Android cuando se complete la conexión:

```
2019-06-06 20:11:32.160 23484-23497/software.amazon.freertos.demo I/FRD: BLE
 connection state changed: 0; new state: BLE_CONNECTED
```

6. Antes de poder transmitir los mensajes, el dispositivo Amazon FreeRTOS y el dispositivo Android negocian la MTU. Debería ver la siguiente salida en logcat:

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD:
 onMTUChanged : 512 status: Success
```

7. El dispositivo se conecta a la aplicación y comienza a enviar mensajes MQTT mediante el proxy MQTT. Para confirmar que el dispositivo puede comunicarse, asegúrese de que el valor de los datos de característica de MQTT_CONTROL cambie a 01:

```
2019-06-06 20:12:28.752 23484-23496/software.amazon.freertos.demo D/FRD: <-<-<-
 Writing to characteristic: MQTT_CONTROL with data: 01
2019-06-06 20:12:28.839 23484-23496/software.amazon.freertos.demo D/FRD:
 onCharacteristicWrite for: MQTT_CONTROL; status: Success; value: 01
```

8. Cuando los dispositivos estén emparejados, aparecerá un mensaje en la consola ESP32. Para activar BLE, pulse y. La demostración no funcionará hasta que realice este paso.

```
E (135538) BT_GATT: GATT_INSUF_AUTHENTICATION: MITM Required
W (135638) BT_L2CAP: l2cble_start_conn_update, the last connection update command
 still pending.
E (135908) BT_SMP: Value for numeric comparison = 391840
15 13588 [InputTask] Numeric comparison:391840
16 13589 [InputTask] Press 'y' to confirm
17 14078 [InputTask] Key accepted
W (146348) BT_SMP: FOR LE SC LTK IS USED INSTEAD OF STK
18 16298 [iot_thread] Connecting to broker...
19 16298 [iot_thread] [INFO ][MQTT][162980] Establishing new MQTT connection.
20 16298 [iot_thread] [INFO ][MQTT][162980] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Waiting for operation completion.
21 16446 [iot_thread] [INFO ][MQTT][164450] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Wait complete with result SUCCESS.
22 16446 [iot_thread] [INFO ][MQTT][164460] New MQTT connection 0x3ffc0ccc
 established.
23 16446 [iot_thread] Connected to broker.
```

Paso 6: Ejecutar el script de actualización OTA

1. Para instalar los requisitos previos, ejecute los siguientes comandos:

```
pip3 install boto3
```

```
pip3 install pathlib
```

2. Aumente la versión de la aplicación FreeRTOS en `demos/include/aws_application_version.h`.
3. Cree un nuevo archivo .bin.
4. Descargue el script de python start_ota.py. Para ver el contenido de ayuda del script, ejecute el siguiente comando en una ventana del terminal:

```
python3 start_ota.py -h
```

Debería ver algo parecido a lo siguiente:

```
usage: start_ota.py [-h] --profile PROFILE [--region REGION]
                    [--account ACCOUNT] [--devicetype DEVICETYPE] --name NAME
                    --role ROLE --s3bucket S3BUCKET --otasigningprofile
                    OTASIGNINGPROFILE --signingcertificateid
                    SIGNINGCERTIFICATEID [--codelocation CODELOCATION]
Script to start OTA update
optional arguments:
-h, --help            show this help message and exit
--profile PROFILE     Profile name created using aws configure
--region REGION       Region
--account ACCOUNT     Account ID
--devicetype DEVICETYPE thing|group
--name NAME           Name of thing/group
--role ROLE           Role for OTA updates
--s3bucket S3BUCKET   S3 bucket to store firmware updates
--otasigningprofile OTASIGNINGPROFILE
                      Signing profile to be created or used
--signingcertificateid SIGNINGCERTIFICATEID
                      certificate id (not arn) to be used
--codelocation CODELOCATION
                      base folder location (can be relative)
```

5. Si usó la plantilla AWS CloudFormation proporcionada para crear recursos, ejecute este comando:

```
python3 start_ota_stream.py --profile otausercf --name esp32-ble --role
 ota_ble_iot_role-sample --s3bucket afr-ble-ota-update-bucket-sample --
otasigningprofile abcd --signingcertificateid certificateid
```

Debería ver el inicio de la actualización en la consola de depuración de ESP32:

```
38 2462 [OTA Task] [prvParseJobDoc] Job was accepted. Attempting to start transfer.
---
49 2867 [OTA Task] [prvIngestDataBlock] Received file block 1, size 1024
50 2867 [OTA Task] [prvIngestDataBlock] Remaining: 1290
51 2894 [OTA Task] [prvIngestDataBlock] Received file block 2, size 1024
52 2894 [OTA Task] [prvIngestDataBlock] Remaining: 1289
53 2921 [OTA Task] [prvIngestDataBlock] Received file block 3, size 1024
54 2921 [OTA Task] [prvIngestDataBlock] Remaining: 1288
55 2952 [OTA Task] [prvIngestDataBlock] Received file block 4, size 1024
56 2953 [OTA Task] [prvIngestDataBlock] Remaining: 1287
57 2959 [iot_thread] State: Active  Received: 5   Queued: 5   Processed: 5
 Dropped: 0
```

6. Cuando finalice la actualización OTA, el dispositivo se reiniciará según lo requiera el proceso de actualización OTA. A continuación, intenta conectarse mediante el firmware actualizado. Si la actualización se ha realizado correctamente, el firmware actualizado se marca como activo y debería ver la versión actualizada en la consola:

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

Aplicación de demostración de sombra de dispositivos AWS IoT

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

Introducción

Esta demostración muestra cómo utilizar la biblioteca de sombra de dispositivo de AWS IoT para conectarse al servicio de sombra de dispositivo de AWS. Utiliza Biblioteca coreMQTT para establecer una conexión MQTT con TLS (autenticación mutua) al agente de AWS IoT MQTT y al analizador de la biblioteca coreJSON para analizar los documentos sombra recibidos del servicio de sombra de AWS. La demostración muestra las operaciones de sombra básicas, por ejemplo, cómo actualizar un documento de sombra y cómo eliminarlo. La demostración también muestra cómo registrar una función de devolución de llamada en la biblioteca coreMQTT para gestionar mensajes como los mensajes de sombra `/update` y `/update/delta` que se envían desde el servicio de sombra de dispositivo de AWS IoT.

Esta demostración solo pretende ser un ejercicio de aprendizaje, ya que la solicitud de actualización del documento de sombra (estado) y la respuesta a la actualización las realiza la misma aplicación. En un escenario de producción realista, una aplicación externa solicitaría una actualización del estado del dispositivo de forma remota, incluso si el dispositivo no está conectado actualmente. El dispositivo confirmará la solicitud de actualización cuando esté conectado.

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.

Funcionalidad

La demostración crea una tarea de aplicación única que incluye una serie de ejemplos en los que se muestran las devoluciones de llamada `/update` y `/update/delta` de sombra para simular el cambio de estado de un dispositivo remoto. Envía una actualización de sombra con el nuevo estado `desired` y espera a que el dispositivo cambie su estado `reported` en respuesta al nuevo estado `desired`. Además, se utiliza una devolución de llamada `/update` de sombra para imprimir los estados de sombra que cambian. Esta demostración también utiliza una conexión MQTT segura con el agente de AWS IoT MQTT y supone que hay un estado `power0n` en la sombra de dispositivo.

La demostración lleva a cabo las siguientes operaciones:

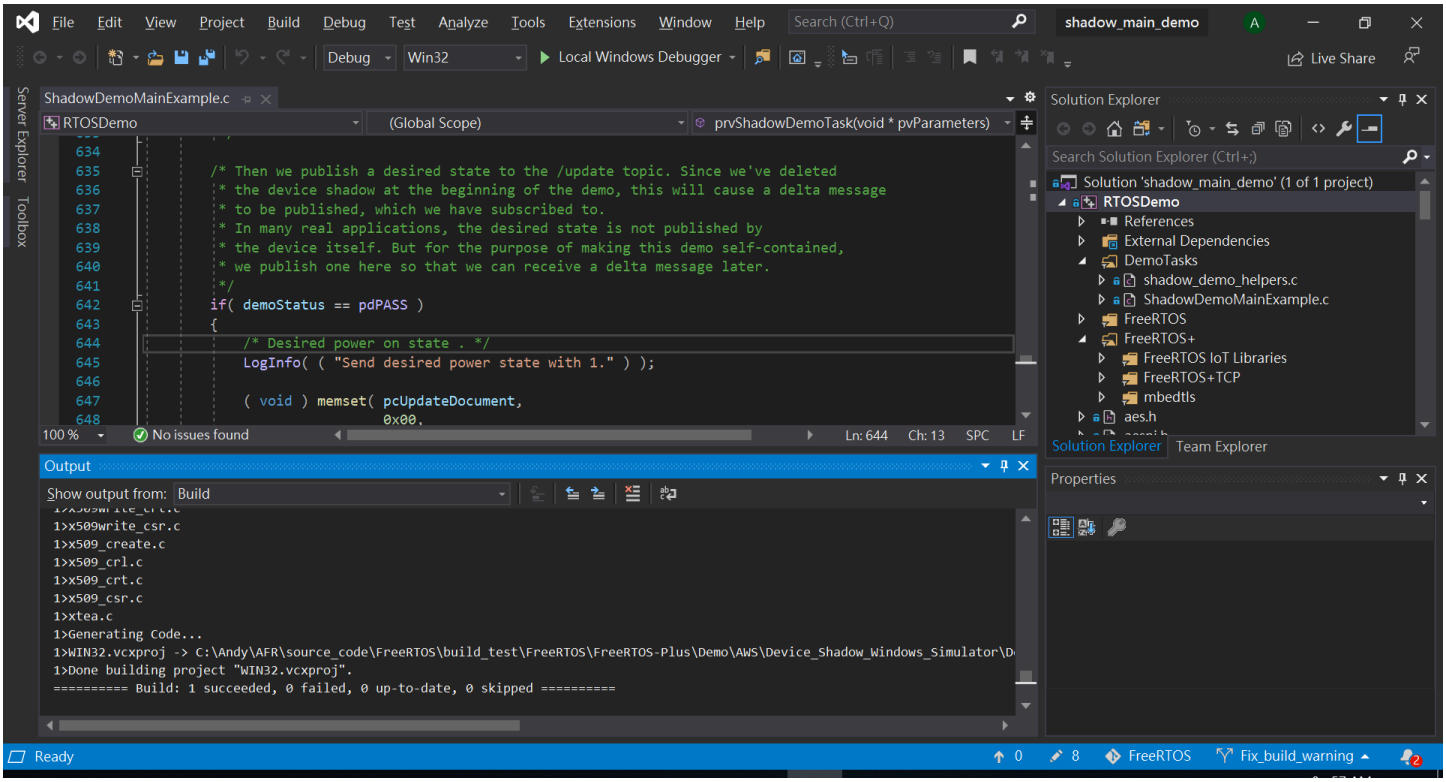
1. Establecea una conexión MQTT mediante las funciones auxiliares de `shadow_demo_helpers.c`.

2. Reúne cadenas de temas de MQTT para las operaciones de sombra de dispositivo mediante las macros definidas por la biblioteca de sombra de dispositivo de AWS IoT.
3. Publica en el tema MQTT utilizado para eliminar una sombra de dispositivo para eliminar cualquier sombra de dispositivo existente.
4. Se suscribe a temas de MQTT correspondientes a `/update/delta`, `/update/accepted` y `/update/rejected` utilizando las funciones auxiliares de `shadow_demo_helpers.c`.
5. Publica el estado deseado de `powerOn` utilizando las funciones auxiliares de `shadow_demo_helpers.c`. Esto provocará que se envíe un mensaje `/update/delta` al dispositivo.
6. Gestiona los mensajes MQTT entrantes en `prvEventCallback` y determina si el mensaje está relacionado con la sombra de dispositivo mediante una función definida en la biblioteca de sombra de dispositivo de AWS IoT (`Shadow_MatchTopic`). Si se trata de un mensaje `/update/delta` de sombra de dispositivo, la función de demostración principal publicará un segundo mensaje para actualizar el estado notificado a `powerOn`. Si recibe un mensaje `/update/accepted`, compruebe que es el mismo `clientToken` que el publicado anteriormente en el mensaje de actualización. Esto marcará el final de la demostración.

```
82 9136 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:641] 83 9136 [ShadowDemo] Send desired power state with 1.84 9136 [ShadowDemo]
85 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 86 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [Shad
owDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:{"version":1,"timestamp":1602751002,"state":{"powerOn":1}
,"metadata":{"powerOn":{"timestamp":1602751002}},"clientToken":"009136"}.90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, ulCurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, ulCurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.102 9296
 [ShadowDemo]
103 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 104 9296 [ShadowDemo] /update/accepted json payload:{"state":{"desired":{"powerOn":1}},"metadata":{"d
esired":{"powerOn":{"timestamp":1602751002}}},"version":1,"timestamp":1602751002,"clientToken":"009136"}.105 9296 [ShadowDemo]
106 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 107 9296 [ShadowDemo] clientToken: 009136108 9296 [ShadowDemo]
109 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 110 9296 [ShadowDemo] receivedToken:9136, clientToken:0
111 9296 [ShadowDemo]
112 9296 [ShadowDemo] [WARN] [SHADOW] [prvUpdateAcceptedHandler:442] 113 9296 [ShadowDemo] The received clientToken=9136 is not identical with the one=0 we sent 114 9296
[ShadowDemo]
115 9696 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:670] 116 9696 [ShadowDemo] Report to the state change: 1117 9696 [ShadowDemo]
118 9856 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 119 9856 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.120 9856
 [ShadowDemo]
121 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 122 9856 [ShadowDemo] /update/accepted json payload:{"state":{"reported":{"powerOn":1}},"metadata":{"
reported":{"powerOn":{"timestamp":1602751003}}},"version":2,"timestamp":1602751003,"clientToken":"009696"}.123 9856 [ShadowDemo]
124 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 125 9856 [ShadowDemo] clientToken: 009696126 9856 [ShadowDemo]
127 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 128 9856 [ShadowDemo] receivedToken:9696, clientToken:9696
129 9856 [ShadowDemo]
130 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:437] 131 9856 [ShadowDemo] Received response from the device shadow. Previously published update with clie
ntToken=9696 has been accepted. 132 9856 [ShadowDemo]
133 10256 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:698] 134 10256 [ShadowDemo] Start to unsubscribe shadow topics and disconnect from MQTT.
135 10256 [ShadowDemo]
136 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:747] 137 12036 [ShadowDemo] Demo completed successfully.138 12036 [ShadowDemo]
139 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:756] 140 12036 [ShadowDemo] Deleting shadow demo task.141 12036 [ShadowDemo]
```

La demostración se encuentra en el archivo *`freertos`*`/demos/device_shadow_for_aws/shadow_demo_main.c` o en GitHub.

En la siguiente captura de pantalla se muestra la salida esperada si la demostración se realiza correctamente.



Conexión al agente MQTT de AWS IoT

Para conectarse al agente AWS IoT MQTT, utilizamos el mismo método que `MQTT_Connect()` en Demostración de la autenticación mutua de coreMQTT.

Eliminación del documento de sombra

Para eliminar el documento de sombra, llame a `xPublishToTopic` con un mensaje vacío y utilice las macros definidas por la biblioteca de sombra de dispositivo de AWS IoT. Utiliza `MQTT_Publish` para publicar en el tema `/delete`. En la siguiente sección de código se muestra cómo se realiza en la función `prvShadowDemoTask`.

```
/* First of all, try to delete any Shadow document in the cloud. */
returnStatus = PublishToTopic( SHADOW_TOPIC_STRING_DELETE( THING_NAME ),
                               SHADOW_TOPIC_LENGTH_DELETE( THING_NAME_LENGTH ),
                               pcUpdateDocument,
                               0U );
```

Suscripción a temas de sombra

Suscríbase a los temas de sombra de dispositivo para recibir notificaciones del agente de AWS IoT sobre cambios de sombra. Los temas de sombra de dispositivo se agrupan mediante macros definidas en la biblioteca de sombra de dispositivo. En la siguiente sección de código se muestra cómo se realiza en la función `prvShadowDemoTask`.

```
/* Then try to subscribe shadow topics. */
if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_DELTA( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_ACCEPTED( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_REJECTED( THING_NAME_LENGTH ) );
}
```

Envío de actualizaciones de sombra

Para enviar una actualización de sombra, la demostración llama a `xPublishToTopic` con un mensaje en formato JSON, utilizando las macros definidas por la biblioteca de sombra de dispositivo. Utiliza `MQTT_Publish` para publicar en el tema `/delete`. En la siguiente sección de código se muestra cómo se realiza en la función `prvShadowDemoTask`.

```
#define SHADOW_REPORTED_JSON    \
    "{"                         \
    "\"state\":{"               \
    "\"reported\":{"            \
```

```
    "\"powerOn\":%01d"          \
    "}"                         \
    "},"                        \
    "\"clientToken\":\"%06lu\"" \
    "}"
snprintf( pcUpdateDocument,
          SHADOW_REPORTED_JSON_LENGTH + 1,
          SHADOW_REPORTED_JSON,
           ( int ) ulCurrentPowerOnState,
           ( long unsigned ) ulClientToken );

xPublishToTopic( SHADOW_TOPIC_STRING_UPDATE( THING_NAME ),
                 SHADOW_TOPIC_LENGTH_UPDATE( THING_NAME_LENGTH ),
                 pcUpdateDocument,
                 ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );
```

Gestión de los mensajes delta de sombra y los mensajes de actualización de sombra

La función de devolución de llamada del usuario, que se registró en la biblioteca de clientes de coreMQTT mediante la función `MQTT_Init`, nos notificará sobre un evento de paquete entrante. Consulte la función de devolución de llamada prvEventCallback en GitHub.

La función de devolución de llamada confirma que el paquete entrante es de tipo `MQTT_PACKET_TYPE_PUBLISH` y utiliza la API de la biblioteca de sombra de dispositivo `Shadow_MatchTopic` para confirmar que el mensaje entrante es un mensaje de sombra.

Si el mensaje entrante es un mensaje de sombra de tipo `ShadowMessageTypeUpdateDelta`, llamamos a prvUpdateDeltaHandler para que gestione este mensaje. El controlador `prvUpdateDeltaHandler` usa la biblioteca coreJSON para analizar el mensaje y obtener el valor delta del estado `powerOn` y lo compara con el estado actual del dispositivo mantenido localmente. Si son diferentes, el estado del dispositivo local se actualiza para reflejar el nuevo valor del estado `powerOn` del documento de sombra.

Si el mensaje entrante es un mensaje de sombra de tipo `ShadowMessageTypeUpdateAccepted`, llamamos a prvUpdateAcceptedHandler para que gestione este mensaje. El controlador `prvUpdateAcceptedHandler` analiza el mensaje mediante la biblioteca coreJSON para obtener el `clientToken`del mensaje. Esta función de controlador comprueba que el token de cliente del mensaje JSON coincide con el token de cliente utilizado por la aplicación. Si no coincide, la función registra un mensaje de advertencia.

Demostración de cliente de eco de sockets seguros

> **Important**
>
> Esta demostración está alojada en el repositorio de Amazon-FreeRTOS, que está en desuso. Recomendamos empezar por aquí al crear un nuevo proyecto. Si ya tiene un proyecto FreeRTOS existente basado en el repositorio Amazon FreeRTOS, ahora obsoleto, consulte Guía de migración del repositorio Github de Amazon-FreeRTOS.

En el ejemplo siguiente, se utiliza una sola tarea de RTOS. El código fuente para este ejemplo se encuentran en `demos/tcp/aws_tcp_echo_client_single_task.c`.

Antes de comenzar, compruebe que ha descargado FreeRTOS en su microcontrolador y que ha creado y ejecutado los proyectos de demostración de FreeRTOS. Puede clonar o descargar FreeRTOS desde GitHub. Consulte el archivo README.md para obtener instrucciones.

Para ejecutar la demostración

> **Note**
>
> Para configurar y ejecutar las demostraciones de FreeRTOS, sigue los pasos que se indican en Introducción a FreeRTOS.
> Las demostraciones de servidor TCP y cliente no se admiten actualmente en los kits de desarrollo Cypress CYW943907AEVAL1F y CYW954907AEVAL1F.

1. Siga las instrucciones que se indican en Configuración del servidor Echo de TLS en la Guía de portabilidad de FreeRTOS.

   Un servidor de eco TLS debe estar en ejecución y a la escucha en el puerto 9000.

   Durante la configuración, debería haber generado cuatro archivos:

   - `client.pem` (certificado de cliente)
   - `client.key`(clave privada de cliente)
   - `server.pem` (certificado de servidor)
   - `server.key` (clave privada de servidor)

2. Utilice la herramienta `tools/certificate_configuration/CertificateConfigurator.html` para copiar el certificado de cliente (`client.pem`) y la clave privada de cliente (`client.key`) en `aws_clientcredential_keys.h`.
3. Abra el archivo `FreeRTOSConfig.h`.
4. Establezca las variables `configECHO_SERVER_ADDR0`, `configECHO_SERVER_ADDR1`, `configECHO_SERVER_ADDR2`, y `configECHO_SERVER_ADDR3` para los cuatro números enteros que componen la dirección IP donde se está ejecutando TLS Echo Server.
5. Establezca la variable `configTCP_ECHO_CLIENT_PORT` en `9000`, el puerto en el que TLS Echo Server está escuchando.
6. Establezca la variable `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` en `1`.
7. Utilice la herramienta `tools/certificate_configuration/PEMfileToCString.html` para copiar el certificado de servidor (`server.pem`) en `cTlsECHO_SERVER_CERTIFICATE_PEM` en el archivo `aws_tcp_echo_client_single_task.c`.
8. Abra *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` y defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

El microcontrolador y el TLS Echo Server deben estar en la misma red. Cuando se inicia la demostración (`main.c`), debería ver un mensaje de registro que indica `Received correct string from echo server`.

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.