aws

Manual do usuário

Gratuito RTOS

Gratuito RTOS: Manual do usuário

Table of Contents

O que é o FreeRTOS?

Desenvolvido em parceria com as principais empresas de chips do mundo ao longo de um período de 15 anos, e agora baixado a cada 170 segundos, o FreeRTOS é um sistema operacional em tempo real (RTOS) líder de mercado para microcontroladores e microprocessadores pequenos. Distribuído livremente sob a licença de código aberto do MIT, o FreeRTOS inclui um kernel e um conjunto crescente de bibliotecas adequadas para uso em todos os setores. O FreeRTOS foi desenvolvido com ênfase na confiabilidade e facilidade de uso.

O FreeRTOS inclui bibliotecas para conectividade, segurança over-the-air e atualizações (OTA). O FreeRTOS também inclui aplicações de demonstração que mostram os recursos do FreeRTOS em placas qualificadas.

O FreeRTOS é um projeto de código aberto. Você pode baixar o código-fonte, contribuir com alterações ou aprimoramentos ou relatar problemas no GitHub site em https://github.com/FreeRTOS/FreeRTOS.

Liberamos o código do FreeRTOS sob a licença de código aberto do MIT para que você possa usá-lo em projetos comerciais e pessoais.

Também apreciamos as contribuições para a documentação do FreeRTOS (Guia do usuário do FreeRTOS, Guia de portabilidade do FreeRTOS e Guia de qualificação do FreeRTOS). Para visualizar a origem do markdown da documentação, consulte https://github.com/awsdocs/aws-freertos-docs. Isso é liberado de acordo com a licença Creative Commons (CC BY-ND).

Download do código-fonte do FreeRTOS

Faça download dos pacotes mais recentes do FreeRTOS e do Long Term Support (LTS) na página Downloads em freertos.org.

Versionamento do FreeRTOS

Bibliotecas individuais usam números de versão no estilo x.y.z, semelhantes ao versionamento semântico. x é o número da versão principal, y o número da versão secundária e, a partir de 2022, z é o número do patch. Antes de 2022, z era um número de lançamento pontual, que exigia que as primeiras bibliotecas LTS tivessem um número de patch no formato "x.y.z LTS Patch 2".

Os pacotes de biblioteca usam números de versão com carimbo de data no estilo aaaamm.x. E aaaa é o ano, mm é o mês e x é um número de sequência opcional que mostra a ordem de lançamento no mês. No caso do pacote LTS, x é um número de patch sequencial para essa versão LTS. As bibliotecas individuais contidas em um pacote são qualquer que fosse a versão mais recente dessa biblioteca naquela data. Para o pacote LTS, é a versão de patch mais recente das bibliotecas LTS originalmente lançada como uma versão LTS naquela data.

Suporte de longo prazo para o FreeRTOS

As versões de suporte de longo prazo (LTS) do FreeRTOS recebem correções de bugs críticos e de segurança (se houver necessidade) por pelo menos dois anos após o lançamento delas. Com essa manutenção contínua, você pode incorporar correções de bugs em todo o ciclo de desenvolvimento e implantação sem a cara interrupção da atualização para novas versões principais de bibliotecas do FreeRTOS.

Com o FreeRTOS LTS, você obtém o conjunto completo de bibliotecas necessárias para compilar produtos IoT incorporados e conectados com segurança. O LTS ajuda a reduzir os custos de manutenção e testes associados à atualização de bibliotecas em dispositivos que já estão em produção.

O FreeRTOS LTS inclui o kernel do FreeRTOS e as bibliotecas de IoT: FreeRTOS+TCP, CoreQTT, CoreHTTP, CorePKCS11, CoreJSON, OTA, Jobs e Device Shadow. AWS IoT AWS IoT AWS IoT Device Defender AWS IoT Para mais informações, consulte Bibliotecas LTS do FreeRTOS.

Plano de manutenção estendida do FreeRTOS

AWS também oferece o FreeRTOS Extended Maintenance Plan (EMP), que fornece patches de segurança e correções críticas de bugs na versão escolhida do FreeRTOS Long Term Support (LTS) por até dez anos adicionais. Com o EMP do FreeRTOS, os dispositivos de longa duração baseados no FreeRTOS podem contar com uma versão que tem estabilidade de recursos e recebe atualizações de segurança por anos. Você recebe notificações sobre os próximos patches nas bibliotecas do FreeRTOS, para assim planejar a implantação de patches de segurança nos dispositivos da Internet das Coisas (IoT).

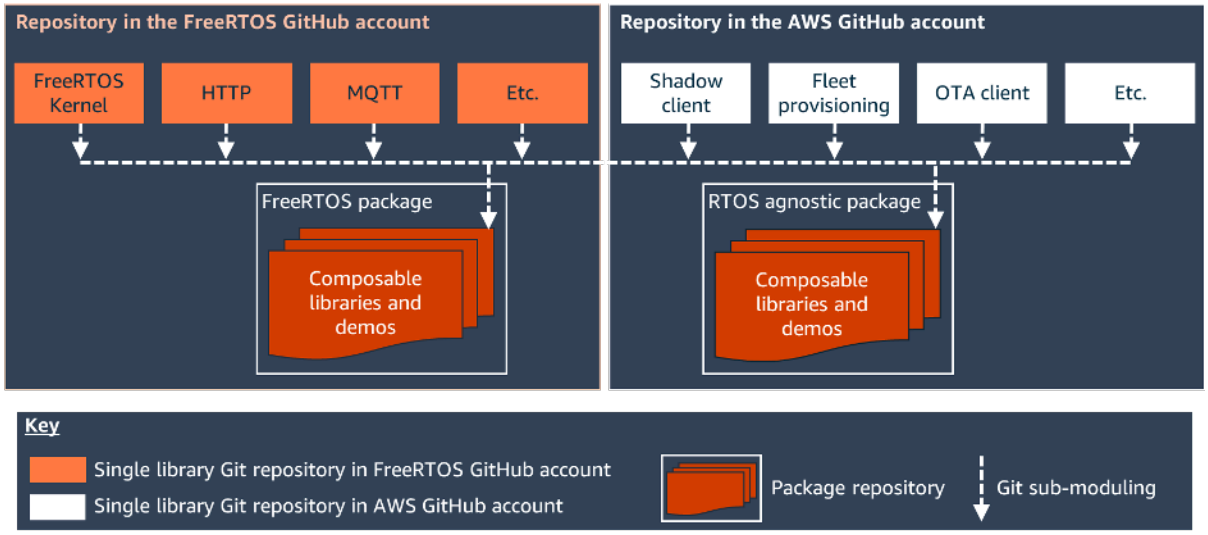
Para saber mais sobre o EMP do FreeRTOS, consulte a página de recursos.

Arquitetura do FreeRTOS

O FreeRTOS contém dois tipos de repositórios, os repositórios de biblioteca única e os de pacotes. Cada repositório de biblioteca contém o código-fonte de uma biblioteca sem projetos de compilação ou exemplos. Os repositórios de pacotes contêm várias bibliotecas e podem conter projetos pré-configurados que demonstram o uso da biblioteca.

Embora os repositórios de pacotes contenham várias bibliotecas, eles não contêm cópias dessas bibliotecas. Em vez disso, os repositórios de pacotes referenciam as bibliotecas que eles contêm como submódulos Git. O uso de submódulos garante a exista uma única fonte confiável para cada biblioteca individual.

Os repositórios git da biblioteca individual são divididos entre duas GitHub organizações. Repositórios contendo bibliotecas específicas do FreeRTOS (como FreeRTOS+TCP) ou bibliotecas genéricas (como o CoreMQTT, que é independente da nuvem porque funciona com qualquer agente MQTT) estão na organização do FreeRTOS. GitHub Repositórios contendo bibliotecas AWS IoT específicas (como o cliente de AWS IoT over-the-air atualização) estão na AWS GitHub organização. O diagrama a seguir explica a estrutura.



Plataformas de hardware qualificadas para o FreeRTOS

As seguintes plataformas de hardware estão qualificados para o FreeRTOS:

- Kit de provisionamento Zero Touch ATECC608A para AWS IoT
- Kit de desenvolvimento Cypress CYW943907AEVAL1F
- Kit de desenvolvimento Cypress CYW954907AEVAL1F

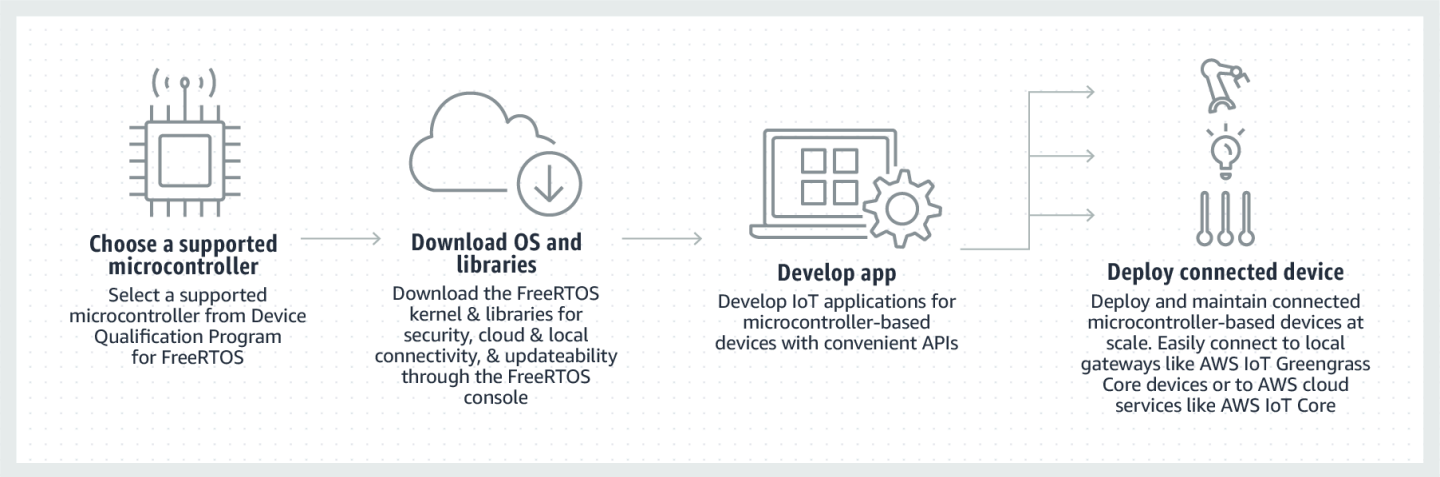
- Kit Cypress CY8CKIT-064S0S2-4343W
- Espressif ESP32- C DevKit
- Espressif ESP-WROVER-KIT
- Espressif ESP-WROOM-32SE
- Espressif ESP32-S2-Saola-1
- Kit de conectividade da IoT Infineon XMC4800
- Kit inicial Marvell MW320 AWS IoT
- Kit inicial Marvell MW322 AWS IoT
- MediaTek Kit de desenvolvimento MT7697hx
- Pacote Microchip Curiosity PIC32MZEF
- Nordic nRF52840-DK
- NuMaker-IoT-M487
- NXP LPC54018 IoT Module
- Solução OPTIGA Trust X Security
- Módulo Iot RSK RX65N da Renesas
- Nó de IoT do Kit Discovery STMicroelectronicsSTM32L4
- CC3220SF-LAUNCHXL da Texas Instruments
- Microsoft Windows 7 ou posterior, com pelo menos um núcleo duplo e uma conexão Ethernet com fio
- Kit de IoT industrial Xilinx Avent MicroZed

Os dispositivos qualificados também estão listados no AWS Partner Device Catalog.

Para obter informações sobre como qualificar um novo dispositivo, consulte o Guia de qualificação do FreeRTOS.

Fluxo de trabalho de desenvolvimento

Para iniciar o desenvolvimento, faça download do FreeRTOS. Você descompacta o pacote e o importa para seu IDE. Em seguida, você pode desenvolver uma aplicação na plataforma de hardware selecionada, fabricar e implantar esses dispositivos usando o processo de desenvolvimento apropriado para seu dispositivo. Os dispositivos implantados podem se conectar ao AWS IoT serviço ou AWS IoT Greengrass como parte de uma solução completa de IoT.



Recursos adicionais

Esses recursos podem ser úteis para você.

- Para obter Documentação adicional do FreeRTOS, consulte freertos.org.
- Para perguntas sobre o FreeRTOS para a equipe de engenharia do FreeRTOS, você pode abrir uma edição na página do FreeRTOS. GitHub
- Para fazer perguntas técnicas sobre o FreeRTOS, consulte os Fóruns da comunidade do FreeRTOS.
- Para obter mais informações sobre como conectar dispositivos a AWS IoT, consulte Provisionamento de dispositivos no Guia do AWS IoT Core desenvolvedor.
- Para obter suporte técnico AWS, consulte o AWS Support Center.
- Para perguntas sobre AWS faturamento, serviços de conta, eventos, abuso ou outros problemas com AWS, consulte a página Fale conosco.

Princípios básicos do kernel do FreeRTOS

O kernel do FreeRTOS é um sistema operacional em tempo real compatível com inúmeras arquiteturas. É ideal para criar aplicativos de microcontroladores incorporados. Ele fornece:

- Um programador multitarefa.
- Várias opções de alocação de memória (inclusive a possibilidade de criar sistemas totalmente alocados estaticamente).
- Primitivos de coordenação entre tarefas, inclusive notificações de tarefas, filas de mensagens, vários tipos de semáforos e buffers de fluxo e de mensagens.
- Suporte para multiprocessamento simétrico (SMP) em microcontroladores de vários núcleos.

O kernel do FreeRTOS nunca realiza operações não determinísticas, como percorrer uma lista vinculada, dentro de uma seção crítica ou interrupção. O kernel do FreeRTOS inclui uma implementação de temporizador de software eficiente que não usa nenhum tempo de CPU, a menos que um temporizador precise de manutenção. Tarefas bloqueadas não exigem manutenção periódica e demorada. Notificações diretas para tarefas permitem a sinalização rápida de tarefas, praticamente sem sobrecarga de RAM. Elas podem ser usadas na maioria dos cenários de sinalização entre tarefas e de interrupção para tarefa.

O kernel do FreeRTOS foi projetado para ser pequeno, simples e fácil de usar. Uma imagem binária típica do kernel do RTOS está no intervalo de 4.000 a 9.000 bytes.

Para obter a documentação mais atualizada sobre o kernel FreeRTOS, consulte FreeRTOS.org. O Freertos.org oferece uma série de tutoriais detalhados e guias sobre o uso do kernel FreeRtos, incluindo um Quick Start Guide (Guia de início rápido) e o documento mais detalhado Mastering the FreeRTOS Real Time Kernel.

Programador de kernel do FreeRTOS

Um aplicativo incorporado que usa um RTOS pode ser estruturado como um conjunto de tarefas independentes. Cada tarefa é executada em seu próprio contexto, sem dependência de outras tarefas. Somente uma tarefa no aplicativo será executada a qualquer momento. O programador do RTOS em tempo real determina quando cada tarefa deve ser executada. Cada tarefa é fornecida com a própria pilha. Quando uma tarefa é trocada para que outra possa ser executada, o contexto de

execução da tarefa é salvo na pilha de tarefas para que ele possa ser restaurado quando a mesma tarefa for trocada de novo posteriormente para retomar sua execução.

Para fornecer um comportamento determinístico em tempo real, o programador de tarefas do FreeRTOS permite que as tarefas recebam prioridades estritas. O RTOS garante que a tarefa de maior prioridade que pode ser executada receba o tempo de processamento. Isso requer o compartilhamento do tempo de processamento entre tarefas de igual prioridade, se elas estiverem prontas para serem executadas simultaneamente. O FreeRTOS também cria uma tarefa inativa que é executada somente quando nenhuma outra tarefa está pronta para ser executada.

Gerenciamento de memória

Essa seção fornece informações sobre a alocação de memória do kernel e o gerenciamento de memória do aplicativo.

Alocação de memória do kernel

O kernel do RTOS precisa de memória RAM toda vez que uma tarefa, fila ou outro objeto do RTOS é criado. A memória RAM pode ser alocada:

- Estaticamente no momento da compilação.
- Dinamicamente a partir do heap do RTOS pelas funções de criação de objeto da API do RTOS.

Quando os objetos do RTOS são criados dinamicamente, o uso da biblioteca `malloc()` e das funções C `free()` padrão nem sempre é apropriado por diversos motivos:

- Elas podem não estar disponíveis em sistemas incorporados.
- Elas ocupam um espaço de código valioso.
- Normalmente, elas não são seguras para o thread.
- Elas não são deterministas.

Por esses motivos, o FreeRTOS mantém a API de alocação de memória em seu nível portátil. A camada portátil está fora dos arquivos de origem que implementam a funcionalidade principal do RTOS, para que você possa fornecer uma implementação específica do aplicativo apropriada para o sistema em tempo real que você está desenvolvendo. Quando o kernel do RTOS exige RAM, ele chama `pvPortMalloc()` em vez de `malloc()`. Quando a memória RAM está sendo liberada, o kernel do RTOS chama `vPortFree()` em vez de `free()`.

Gerenciamento de memória de aplicativos

Quando os aplicativos precisam de memória, eles podem alocá-la do heap do FreeRTOS. O FreeRTOS oferece vários esquemas de gerenciamento de heap que variam em termos de complexidade e recursos. Você também pode fornecer sua própria implementação de heap.

O kernel do FreeRTOS inclui cinco implementações de heap:

`heap_1`

É a implementação mais simples. Não permite que a memória seja liberada.

`heap_2`

Permite que a memória seja liberada, mas não une blocos livres adjacentes.

`heap_3`

Encapsula o `malloc()` e o `free()` padrão para segurança de threads.

`heap_4`

Une blocos livres adjacentes para evitar a fragmentação. Inclui uma opção de posicionamento de endereço absoluto.

`heap_5`

É semelhante a heap_4. Pode abranger o heap em várias áreas de memória não adjacentes.

Coordenação entre tarefas

Esta seção contém informações sobre primitivos do FreeRTOS.

Filas

As filas são a principal forma de comunicação entre tarefas. Elas podem ser usadas para enviar mensagens entre tarefas e entre interrupções e tarefas. Na maioria dos casos, elas são usadas como buffers FIFO (primeiro a entrar, primeiro a sair) seguros para threads com novos dados sendo enviados para o final da fila. (Os dados também podem ser enviados para a frente da fila.) As mensagens são enviadas por meio de filas por cópia, o que significa que os dados (que podem ser um ponteiro para buffers maiores) são copiados na fila, em vez de simplesmente armazenar uma referência aos dados.

As APIs de fila permitem que um tempo de bloqueio seja especificado. Quando uma tarefa tenta ler a partir de uma fila vazia, a tarefa é colocada no estado Bloqueada até que os dados se tornem disponíveis na fila ou que o tempo de bloqueio termine. As tarefas no estado Bloqueadas não consomem nenhum tempo de CPU, permitindo que outras tarefas sejam executadas. Da mesma forma, quando uma tarefa tenta gravar em uma fila cheia, a tarefa é colocada no estado Bloqueada até que o espaço seja disponibilizado na fila ou o tempo de bloqueio termine. Se mais de uma tarefa bloquear na mesma fila, a tarefa com a prioridade mais alta será desbloqueada primeiro.

Outros primitivos do FreeRTOS, como notificações diretas para tarefas e buffers de fluxo e de mensagens, oferecem alternativas leves para filas em muitos cenários de projeto comuns.

Semáforos e mutexes

O kernel do FreeRTOS fornece semáforos binários, semáforos de contagem e mutexes para fins de sincronização e exclusão mútua.

Semáforos binários só podem ter dois valores. Eles são uma boa opção para implementar a sincronização (entre tarefas ou entre tarefas e uma interrupção). Os semáforos de contagem podem usar mais de dois valores. Eles permitem que muitas tarefas compartilhem recursos ou executem operações de sincronização mais complexas.

Mutexes são semáforos binários que incluem um mecanismo de herança de prioridade. Isso significa que, se uma tarefa de alta prioridade for bloqueada ao tentar obter um mutex que esteja retido no momento por uma tarefa de prioridade mais baixa, a prioridade da tarefa que contém o token é temporariamente aumentada para o nível da tarefa de bloqueio. Esse mecanismo é projetado para garantir que a tarefa de prioridade mais alta seja mantida no estado Bloqueada pelo menor tempo possível, para minimizar a inversão de prioridade que ocorreu.

Notificações diretas para tarefas

As notificações de tarefas permitem que as tarefas interajam com outras tarefas e sincronizem com rotinas de serviço de interrupção (ISRs), sem a necessidade de um objeto de comunicação separado, como um semáforo. Cada tarefa do RTOS possui um valor de notificação de 32 bits que é usado para armazenar o conteúdo da notificação, se houver. Uma notificação de tarefa do RTOS é um evento enviado diretamente a uma tarefa que pode desbloquear a tarefa de recebimento e, opcionalmente, atualizar o valor de notificação da tarefa de recebimento.

As notificações de tarefas do RTOS podem ser usadas como uma alternativa mais rápida e leve aos semáforos binários e de contagem e, em alguns casos, às filas. Notificações de tarefas têm

vantagens de velocidade e espaço de RAM em relação a outros recursos do FreeRTOS que podem ser usados para executar funcionalidades equivalentes. No entanto, as notificações de tarefas só podem ser usadas quando houver apenas uma tarefa que possa ser o destinatário do evento.

Buffers de fluxo

Os buffers de fluxo permitem que um fluxo de bytes seja transmitido de uma rotina de serviço de interrupção para uma tarefa ou de uma tarefa para outra. Um fluxo de bytes pode ser de tamanho arbitrário e não tem necessariamente um começo ou um fim. Qualquer número de bytes pode ser gravado de uma só vez e lido ao mesmo tempo. Você ativa a funcionalidade de buffer de fluxo incluindo o arquivo de origem `stream_buffer.c` em seu projeto.

Os buffers de fluxo presumem que há apenas uma tarefa ou interrupção que grava no buffer (o gravador) e apenas uma tarefa ou interrupção que lê a partir do buffer (o leitor). É seguro que o gravador e o leitor sejam tarefas ou rotinas de serviço de interrupção diferentes, mas não é seguro ter vários gravadores ou leitores.

A implementação do buffer de fluxo usa as notificações diretas para tarefas. Portanto, chamar uma API de buffer de fluxo que coloca a tarefa de chamada no estado Bloqueada pode alterar o estado e o valor de notificação da tarefa de chamada.

Envio de dados

`xStreamBufferSend()` é usado para enviar dados a um buffer de fluxo em uma tarefa. `xStreamBufferSendFromISR()` é usado para enviar dados a um buffer de fluxo em uma rotina de serviço de interrupção (ISR).

`xStreamBufferSend()` permite que um tempo de bloqueio seja especificado. Se `xStreamBufferSend()` for chamado com um tempo de bloqueio diferente de zero para gravar em um buffer de fluxo e o buffer estiver cheio, a tarefa será colocada no estado Bloqueada até que o espaço seja disponibilizado ou o tempo de bloqueio expire.

`sbSEND_COMPLETED()` e `sbSEND_COMPLETED_FROM_ISR()` são macros que são chamadas (internamente pela API do FreeRTOS) quando os dados são gravados em um buffer de fluxo. É utilizado o identificador do buffer de fluxo que foi atualizado. As duas macros verificam se há uma tarefa bloqueada no buffer de fluxo aguardando dados e, se esse for o caso, removem a tarefa do estado Bloqueada.

Você pode alterar esse comportamento padrão fornecendo sua própria implementação de `sbSEND_COMPLETED()` em `FreeRTOSConfig.h` Isso é útil quando um buffer de fluxo é

usado para transmitir dados entre núcleos em um processador de vários núcleos. Nesse cenário, `sbSEND_COMPLETED()` pode ser implementado para gerar uma interrupção no outro núcleo da CPU, em seguida, a rotina de serviço da interrupção poderá usar a API `xStreamBufferSendCompletedFromISR()` para verificar e, se necessário, desbloquear uma tarefa que esteja aguardando os dados.

Recebimento de dados

`xStreamBufferReceive()` é usado para ler dados de um buffer de fluxo em uma tarefa. `xStreamBufferReceiveFromISR()` é usado para ler dados de um buffer de fluxo em uma rotina de serviço de interrupção (ISR).

`xStreamBufferReceive()` permite que um tempo de bloqueio seja especificado. Se `xStreamBufferReceive()` for chamado com um tempo de bloqueio diferente de zero para ler de um buffer de fluxo e o buffer estiver vazio, a tarefa será colocada no estado Bloqueada até que uma quantidade especificada de dados seja disponibilizada no buffer de fluxo ou o tempo de bloqueio expire.

A quantidade de dados que deve estar no buffer de fluxo antes de uma tarefa ser desbloqueada é chamada de nível de ativação do buffer de fluxo. Uma tarefa bloqueada com um nível de ativação de 10 é desbloqueada quando pelo menos 10 bytes são gravados no buffer ou o tempo de bloqueio da tarefa expira. Se o tempo de bloqueio de uma tarefa de leitura expirar antes do nível de ativação ser atingido, a tarefa receberá todos os dados gravados no buffer. O nível de ativação de uma tarefa deve ser definido para um valor entre 1 e o tamanho do buffer de fluxo. O nível de ativação de um buffer de fluxo é definido quando `xStreamBufferCreate()` é chamado. É possível alterá-lo chamando `xStreamBufferSetTriggerLevel()`.

`sbRECEIVE_COMPLETED()` e `sbRECEIVE_COMPLETED_FROM_ISR()` são macros que são chamadas (internamente pela API do FreeRTOS) quando os dados são lidos de um buffer de fluxo. As macros verificam se há uma tarefa bloqueada no buffer de fluxo aguardando que o espaço fique disponível dentro do buffer e, se esse for o caso, removem a tarefa do estado Bloqueada. Você pode alterar o comportamento padrão de `sbRECEIVE_COMPLETED()` fornecendo uma implementação alternativa no `FreeRTOSConfig.h`.

Buffers de mensagens

Os buffers de mensagens permitem que mensagens discretas de tamanho variável sejam transmitidas de uma rotina de serviço de interrupção para uma tarefa ou de uma tarefa para outra. Por exemplo, mensagens com 10, 20 e 123 bytes de tamanho podem ser gravadas e lidas a partir do

mesmo buffer de mensagens. Uma mensagem de 10 bytes só pode ser lida como uma mensagem de 10 bytes, não como bytes individuais. Os buffers de mensagens são criados na implementação do buffer de fluxo. É possível habilitar a funcionalidade do buffer de mensagens incluindo o arquivo de origem `stream_buffer.c` em seu projeto.

Os buffers de mensagens presumem que há apenas uma tarefa ou interrupção que grava no buffer (o gravador) e apenas uma tarefa ou interrupção que lê a partir do buffer (o leitor). É seguro que o gravador e o leitor sejam tarefas ou rotinas de serviço de interrupção diferentes, mas não é seguro ter vários gravadores ou leitores.

A implementação do buffer de mensagens usa as notificações diretas para tarefas. Portanto, chamar uma API de buffer de fluxo que coloca a tarefa de chamada no estado Bloqueada pode alterar o estado e o valor de notificação da tarefa de chamada.

Para permitir que os buffers de mensagens manipulem mensagens de tamanho variável, o tamanho de cada mensagem é gravado no buffer de mensagens antes da própria mensagem. O tamanho é armazenado em uma variável do tipo `size_t`, que normalmente é de 4 bytes em uma arquitetura de 32 bytes. Portanto, gravar uma mensagem de 10 bytes em um buffer de mensagens consome na verdade 14 bytes de espaço em buffer. Da mesma forma, gravar uma mensagem de 100 bytes em um buffer de mensagens usa na verdade 104 bytes de espaço em buffer.

Envio de dados

`xMessageBufferSend()` é usado para enviar dados a um buffer de mensagens a partir de uma tarefa. `xMessageBufferSendFromISR()` é usado para enviar dados a um buffer de mensagens a partir de uma rotina de serviço de interrupção (ISR).

`xMessageBufferSend()` permite que um tempo de bloqueio seja especificado. Se `xMessageBufferSend()` for chamado com um tempo de bloqueio diferente de zero para gravar em um buffer de mensagens e o buffer estiver cheio, a tarefa será colocada no estado Bloqueada até que o espaço seja disponibilizado no buffer de mensagens ou o tempo de bloqueio expire.

`sbSEND_COMPLETED()` e `sbSEND_COMPLETED_FROM_ISR()` são macros que são chamadas (internamente pela API do FreeRTOS) quando os dados são gravados em um buffer de fluxo. É utilizado um único parâmetro, que é o identificador do buffer de fluxo que foi atualizado. As duas macros verificam se há uma tarefa bloqueada no buffer de fluxo aguardando dados e, se esse for o caso, elas removem a tarefa do estado Bloqueada.

Você pode alterar esse comportamento padrão fornecendo sua própria implementação de `sbSEND_COMPLETED()` em `FreeRTOSConfig.h` Isso é útil quando um buffer de fluxo é

usado para transmitir dados entre núcleos em um processador de vários núcleos. Nesse cenário, `sbSEND_COMPLETED()` pode ser implementado para gerar uma interrupção no outro núcleo da CPU, em seguida, a rotina de serviço da interrupção poderá usar a API `xStreamBufferSendCompletedFromISR()` para verificar e, se necessário, desbloquear uma tarefa que estava aguardando os dados.

Recebimento de dados

`xMessageBufferReceive()` é usado para ler dados de um buffer de mensagens em uma tarefa. `xMessageBufferReceiveFromISR()` é usado para ler dados de um buffer de mensagens em uma rotina de serviço de interrupção (ISR). `xMessageBufferReceive()` permite que um tempo de bloqueio seja especificado. Se `xMessageBufferReceive()` for chamado com um tempo de bloqueio diferente de zero para ler a partir de um buffer de mensagens e o buffer estiver vazio, a tarefa será colocada no estado Bloqueada até que os dados sejam disponibilizados ou o tempo de bloqueio expire.

`sbRECEIVE_COMPLETED()` e `sbRECEIVE_COMPLETED_FROM_ISR()` são macros que são chamadas (internamente pela API do FreeRTOS) quando os dados são lidos de um buffer de fluxo. As macros verificam se há uma tarefa bloqueada no buffer de fluxo aguardando que o espaço fique disponível dentro do buffer e, se esse for o caso, removem a tarefa do estado Bloqueada. Você pode alterar o comportamento padrão de `sbRECEIVE_COMPLETED()` fornecendo uma implementação alternativa no `FreeRTOSConfig.h`.

Suporte ao multiprocessamento simétrico (SMP)

O suporte ao SMP no kernel do FreeRTOS permite que uma instância kernel do FreeRTOS agende tarefas em vários núcleos de processador idênticos. As arquiteturas principais devem ser idênticas e compartilhar a mesma memória.

Modificação de aplicações para usar o kernel FreeRTOS-SMP

A API do FreeRTOS permanece substancialmente a mesma entre as versões de núcleo único e de SMP, exceto essas APIs adicionais. Portanto, uma aplicação escrita para a versão de núcleo único do FreeRTOS deve ser compilado com a versão SMP com o mínimo ou nenhum esforço. No entanto, pode haver alguns problemas funcionais, porque algumas suposições que eram verdadeiras para aplicações de núcleo único podem não ser mais verdadeiras para aplicações de vários núcleos.

Uma suposição comum é que uma tarefa de menor prioridade não pode ser executada enquanto uma tarefa de maior prioridade está em execução. Embora isso fosse verdade em um sistema de

núcleo único, não é mais verdade para sistemas de vários núcleos porque várias tarefas podem ser executadas simultaneamente. Se a aplicação se basear em prioridades de tarefas relativas para fornecer exclusão mútua, ela poderá observar resultados inesperados em um ambiente com vários núcleos.

Outra suposição comum é que os ISRs não podem ser executados simultaneamente entre si ou com outras tarefas. Isso não é mais verdade em um ambiente com vários núcleos. O criador da aplicação precisa garantir a exclusão mútua adequada ao acessar dados compartilhados entre tarefas e ISRs.

Temporizadores de software

Um temporizador de software permite que uma função seja executada em um horário definido no futuro. A função executada pelo temporizador é chamada de função de retorno de chamada do temporizador. O tempo entre um temporizador ser iniciado e sua função de retorno de chamada ser executada é chamado de período do temporizador. O kernel do FreeRTOS fornece uma implementação de temporizador de software eficiente porque:

- Não executa funções de retorno de chamada do temporizador a partir de um contexto de interrupção.
- Não consome nenhum tempo de processamento, a menos que um temporizador tenha expirado.
- Não adiciona nenhuma sobrecarga de processamento à interrupção do tique.
- Não percorre estruturas de lista de links enquanto as interrupções estão desativadas.

Suporte para baixo consumo

Como a maioria dos sistemas operacionais incorporados, o kernel do FreeRTOS usa um temporizador de hardware para gerar interrupções de tique periódicas, que são usadas para medir o tempo. A economia de energia de implementações de temporizador de hardware regulares é limitada pela necessidade de sair periodicamente do estado de baixo consumo e, em seguida, entrar novamente para processar as interrupções de tique. Se a frequência da interrupção de tique for muito alta, a energia e o tempo consumidos ao entrar e sair de um estado de baixo consumo para cada tique superam quaisquer ganhos potenciais de economia de energia, exceto dos modos de economia de energia mais leves.

Para resolver essa limitação, o FreeRTOS inclui um modo de temporizador sem tiques para aplicativos de baixo consumo. O modo de inatividade sem tique do FreeRTOS para a interrupção periódica de tique durante períodos inativos (períodos em que não há tarefas de aplicativo que

podem ser executadas) e, em seguida, faz um ajuste de correção no valor de contagem de tiques do RTOS quando a interrupção de tique é reiniciada. Parar a interrupção de tique permite que o microcontrolador permaneça em um estado de economia de energia profunda até que ocorra uma interrupção ou seja o momento de o kernel do RTOS fazer a transição de uma tarefa para o estado pronto.

Configuração do kernel

Você pode configurar o kernel do FreeRTOS para uma placa e aplicativo específicos com o arquivo de cabeçalho `FreeRTOSConfig.h`. Todo aplicativo criado no kernel deve ter um arquivo de cabeçalho `FreeRTOSConfig.h` em seu caminho de inclusão do pré-processador. `FreeRTOSConfig.h` é específico do aplicativo e deve ser colocado sob um diretório de aplicativo e não em um dos diretórios de código-fonte do kernel do FreeRTOS.

Os arquivos `FreeRTOSConfig.h` das aplicações de demonstração e teste do FreeRTOS estão localizados em *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/FreeRTOSConfig.h` e *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_tests/config_files/FreeRTOSConfig.h`.

Para obter uma lista dos parâmetros de configuração disponíveis para especificar em `FreeRTOSConfig.h`, consulte FreeRTOS.org.

SDK de dispositivo da AWS IoT para C incorporado

> **Note**
>
> Esse SDK é destinado ao uso por desenvolvedores de software incorporados experientes.

O AWS IoT Device SDK para C incorporado (C-SDK) é um conjunto de arquivos de origem C na licença de código-aberto do MIT que pode ser usado em aplicações incorporadas para conectar dispositivos IoT à AWS IoT Core com segurança. Isso inclui um cliente MQTT, cliente HTTP, analisador JSON, Device Shadow do AWS IoT, Trabalhos do AWS IoT, Provisionamento de frota do AWS IoT e bibliotecas AWS IoT Device Defender. Esse SDK é distribuído na forma de origem e pode ser compilado no firmware do cliente junto ao código da aplicação, outras bibliotecas e um sistema operacional (SO) de sua preferência.

Em geral, o AWS IoT Device SDK para C incorporado destina-se a dispositivos com restrição de recursos que exigem um tempo de execução de linguagem C otimizado. É possível usar o SDK em qualquer sistema operacional e hospedá-lo em qualquer tipo de processador (p. ex., MCUs e MPUs). No entanto, se seus dispositivos tiverem memória e recursos de processamento suficientes, recomendamos que você use um dos AWS IoT Device SDKs de ordem superior.

Para obter mais informações, consulte as informações a seguir.

- AWS IoT Device SDK para C incorporado
- SDK de dispositivo AWS IoT para C incorporado no GitHub
- Arquivo leiame do SDK de dispositivo da AWS IoT para C incorporado
- Exemplos de AWS IoT Device SDK para C incorporado

E/S comuns

As APIs de E/S comuns atuam como camadas de abstração de hardware (HAL) que fornecem uma interface comum entre drivers e código de aplicativo de nível superior. As E/S comuns do FreeRTOS fornecem um conjunto de APIs padrão para acessar dispositivos seriais comuns em placas de referência compatíveis; as implementações dessas APIs não estão incluídas. Essas APIs comuns se comunicam e interagem com esses periféricos e permitem que seu código funcione entre plataformas. Sem as E/S comuns, escrever código para que funcione com dispositivos de baixo nível é uma tarefa específica do fornecedor Sillicon.

> **Note**
>
> O FreeRTOS não exige implementações das APIs de E/S comuns para funcionar, mas tentará usar as APIs de E/S comuns como uma forma de interagir com os periféricos específicos em uma placa baseada em microcontrolador, em vez de APIs específicas do fornecedor.

Em geral, os drivers de dispositivo são independentes do sistema operacional subjacente e são específicos de uma determinada configuração de hardware. A HAL abstrai os detalhes de como um driver específico funciona e fornece uma API uniforme para controlar esses dispositivos. Você pode usar as mesmas APIs para acessar vários drivers de dispositivo em placas de referência baseadas em múltiplos microcontroladores (MCU).

Bibliotecas

Atualmente, o FreeRTOS fornece duas bibliotecas de E/S comuns: a E/S comum - básica e a E/S comum - BLE.

E/S comum - básica

Visão geral

A E/S comum - básica fornece APIs que lidam com periféricos e funções básicas de E/S que você pode encontrar em placas baseadas em MCU. O repositório da E/S comum - básica está disponível no GitHub.

Periféricos compatíveis

- ADC
- GPIO
- I2C
- PWM
- SPI
- UART
- WatchDog
- Flash
- ETC
- EFUSE
- Reinicializações
- I2
- Contador de desempenho
- Informações sobre a plataforma de hardware

Recursos compatíveis

- Leitura/gravação síncrona

  A função não retorna até que a quantidade solicitada de dados seja transferida.

- Leitura/gravação assíncrona

  A função retorna imediatamente e a transferência de dados ocorre de forma assíncrona. Quando a ação for concluída, um retorno de chamada de usuário registrado é chamado.

Específico do periférico

- I2C

  Combina várias operações em uma transação. Usado para realizar as ações de gravação e leitura em uma transação.

- SPI

  Transfere dados entre primário e secundário, o que significa que a gravação e a leitura ocorrem simultaneamente.

Referência de API

Para obter uma referência completa de API, consulte a Referência de API de E/S comum - básica.

E/S comum - BLE

Visão geral

A E/S comum - BLE fornece abstração da pilha Bluetooth Low Energy do fabricante. Ela fornece as seguintes interfaces que podem ser usadas para controlar o dispositivo e realizar operações GAP e GATT. O repositório da E/S comum - BLE está disponível no GitHub.

Gerenciador de dispositivos Bluetooth:

Isso fornece uma interface para controlar o dispositivo Bluetooth, realizar operações de descoberta de dispositivos e outras tarefas relacionadas à conectividade.

Gerenciador de adaptadores BLE:

Isso fornece uma interface para as funções da API GAP que são específicas do BLE.

Gerenciador de adaptadores Bluetooth clássicos:

Isso fornece uma interface para controlar as funcionalidades clássicas de BT de um dispositivo.

Servidor GATT:

Isso fornece uma interface para usar o atributo de servidor Bluetooth GATT.

Cliente GATT:

Isso fornece uma interface para usar o atributo de cliente Bluetooth GATT.

Interface de conexão A2DP:

Isso fornece uma interface do perfil de origem A2DP para o dispositivo local.

Referência de API

Para obter uma referência completa de API, consulte a Referência de API de E/S comum - BLE.

E/S comum para o Amazon Common Software

As APIs de E/S comuns fazem parte das implementações exigidas pelo Amazon Common Software para dispositivos, especificamente para serem implementadas em um kit de portabilidade de dispositivos (DPK) do fornecedor.

O que é o ACS?

O Amazon Common Software (ACS) para dispositivos é um software que agiliza a integração de SDKs do Amazon Device em seus dispositivos. O ACS fornece uma camada de integração de API unificada, componentes pré-validados e com uso eficiente de memória para funções comuns, como conectividade, kit de portabilidade de dispositivos (DPK) e pacotes de testes de várias camadas.

Programa de qualificação

O programa de qualificação do Amazon Common Software para dispositivos verifica se uma versão DPK (kit de portabilidade de dispositivos) do ACS que é executada em uma placa de desenvolvimento específica baseada em microcontrolador é compatível com as práticas recomendadas publicadas pelo programa e se ela é robusta o suficiente para passar nos testes exigidos pelo ACS especificados pelo programa de qualificação.

Os fornecedores qualificados neste programa estão listados na página Fornecedores de chipsets do ACS.

Para obter mais informações sobre a qualificação do dispositivo, consulte Dispositivos para o ACS.

Conceitos básicos do FreeRTOS

Tópicos

Conceitos básicos do AWS IoT e do FreeRTOS usando conexão rápida

Para explorar o AWS IoT rapidamente, comece com as Demonstrações de conexão rápida da AWS. As demonstrações de conexão rápida são simples de configurar e conectar uma placa qualificada pelo FreeRTOS e fornecida por um parceiro no AWS IoT.

Siga o tutorial de Começar a usar o AWS IoT para entender melhor o AWS IoT e o console do AWS IoT. Você pode modificar o código-fonte de demonstração fornecido com as demonstrações de conexão rápida usando o sistema de compilação e as ferramentas da placa escolhida para se conectar à sua conta da AWS. Agora, o fluxo de dados do console do AWS IoT está visível em sua conta.

Explore bibliotecas do FreeRTOS

Depois de entender como um dispositivo IoT e o AWS IoT trabalham juntos, você pode começar a explorar as Bibliotecas do FreeRTOS e as Bibliotecas de suporte de longo prazo (LTS).

Algumas bibliotecas comumente usadas para dispositivos AWS IoT baseados em FreeRTOS são:

- Kernel do FreeRTOS
- coreMQTT
- Sem fios do AWS IoT

Visite freertos.org para obter demonstrações e documentações técnicas específicas da biblioteca.

Entenda como compilar um produto do AWS IoT seguro e robusto

Consulte as Integrações AWS IoT do FreeRTOS em destaque para conhecer as práticas recomendadas para tornar o software de dispositivos IoT mais seguro e robusto. Essas integrações IoT do FreeRTOS foram criadas para melhorar a segurança usando uma combinação do software FreeRTOS e uma placa fornecida por um parceiro com atributos de segurança de hardware. Use isso na produção como está ou como modelo para os próprios projetos.

Desenvolver produto de aplicativo do AWS IoT

Siga estas etapas para criar um projeto de aplicativo para seu produto do AWS IoT:

1. Baixe a versão mais recente do FreeRTOS ou do suporte de longo prazo (LTS) em freertos.org ou clone do repositório GitHub FreeRTOS-LTS. Você também pode integrar as bibliotecas FreeRTOS necessárias em seu projeto a partir da cadeia de ferramentas do fornecedor do MCU, se disponível.
2. Siga o Guia de portabilidade do FreeRTOS para criar um projeto, configurar o ambiente de desenvolvimento e integrar as bibliotecas do FreeRTOS ao seu projeto. Use o repositório FreeRTOS-Libraries-Integration-Tests do GitHub para validar a portabilidade.

AWS IoT Device Tester para FreeRTOS

O IDT para FreeRTOS é uma ferramenta para qualificar a taxa de throughput de dados no sistema operacional do FreeRTOS. O testador de dispositivos (IDT) abre primeiro uma conexão USB ou UART com um dispositivo. Em seguida, ele exibe uma imagem dos FreeRTOS configurado para testar a funcionalidade do dispositivo sob várias condições. Os pacotes do AWS IoT Device Tester são extensíveis e o IDT é usado para orquestração de testes do cliente do AWS IoT.

O IDT para FreeRTOS é executado em um computador host (Windows, macOS ou Linux) que está conectado à placa a ser testada. O IDT configura e orquestra casos de teste e reúne resultados. Ele também fornece uma interface de linha de comando para gerenciar a execução do teste.

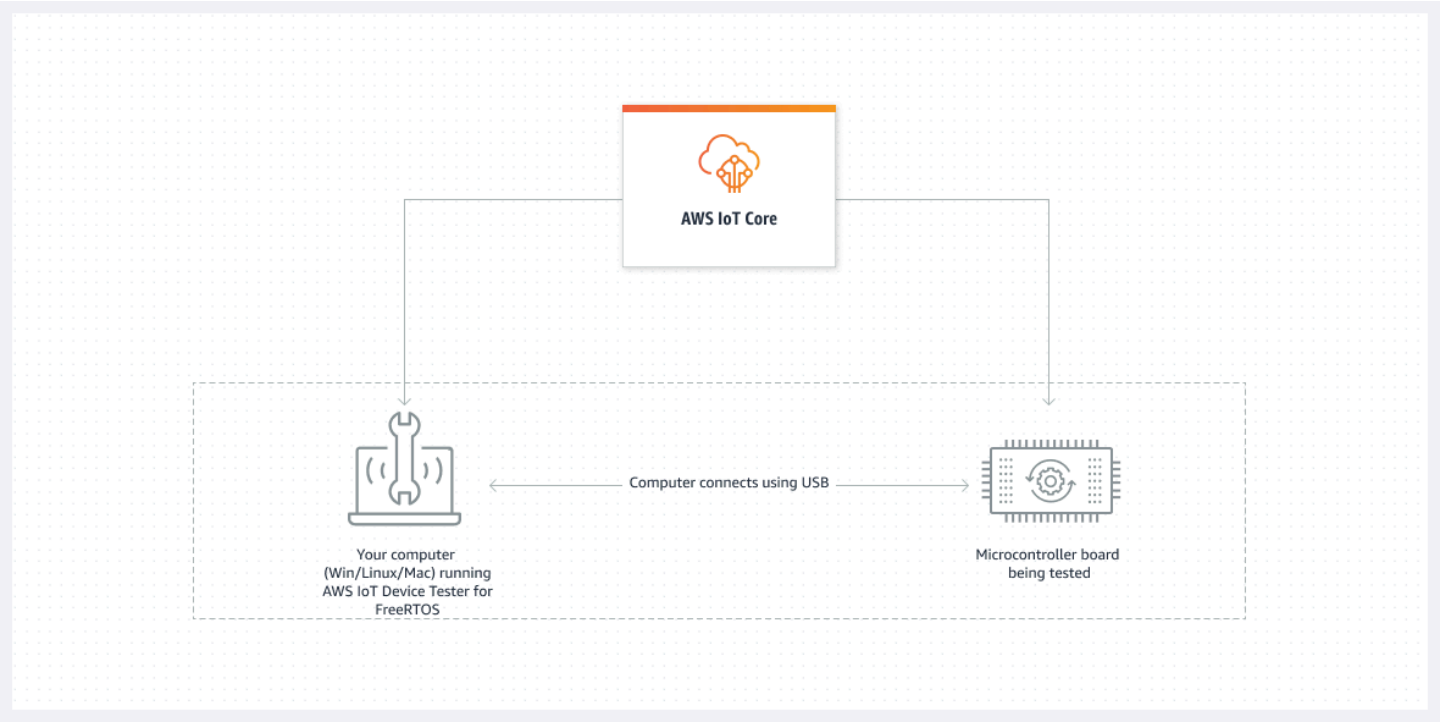
Pacote de qualificação do FreeRTOS

O IDT para FreeRTOS verifica a porta do FreeRTOS em seu microcontrolador e se ela pode efetivamente se comunicar de maneira confiável e segura com o AWS IoT. Especificamente, ele verifica se as interfaces da camada de portabilidade para as bibliotecas do FreeRTOS estão implementadas corretamente. Ele também executa testes de ponta a ponta com o AWS IoT Core. Por exemplo, verifica se a placa pode enviar e receber mensagens MQTT e processá-las corretamente.

O pacote de qualificação do FreeRTOS (FRQ) 2.0 usa casos de testes da FreeRTOS-Libraries-Integration-Tests e do Device Advisor definidos no Guia de qualificação do FreeRTOS.

O IDT for FreeRTOS gera relatórios de teste que você pode enviar ao Rede de Parceiros da AWS (APN) para inclusão de seus dispositivos FreeRTOS no AWS Partner Device Catalog. Para obter mais informações, consulte AWS Device Qualification Program.

O diagrama a seguir mostra a configuração da infraestrutura de teste para a qualificação do FreeRTOS.



O IDT para FreeRTOS organiza testes em pacotes de testes e grupos de testes:

- Um pacote de testes é o conjunto de grupos de teste usado para verificar se um dispositivo funciona com versões específicas do FreeRTOS.
- Um grupo de testes é o conjunto de casos de teste individuais relacionados a um atributo específico, como sistemas de mensagens BLE e MQTT.

Para obter mais informações, consulte Versões do conjunto de testes.

Pacotes de teste personalizados

O IDT para FreeRTOS combina uma configuração padronizada e um formato de resultado com um ambiente de pacotes de teste. Este ambiente permite que você desenvolva pacotes de teste personalizados para seus dispositivos e software de dispositivos. Você pode adicionar testes personalizados a própria validação interna ou fornecê-los aos seus clientes para verificação de dispositivo.

A forma como os pacotes de teste personalizados são configurados determina as configurações que devem ser fornecidas aos usuários para executar seus pacotes de teste personalizados. Para obter mais informações, consulte Usar o IDT para desenvolver e executar os próprios pacotes de testes.

Versões compatíveis do AWS IoT Device Tester para FreeRTOS

Este tópico lista as versões compatíveis do AWS IoT Device Tester para FreeRTOS. Como prática recomendada, é aconselhável usar a versão mais recente do IDT para FreeRTOS que seja compatível com a sua versão de destino do FreeRTOS. Cada versão do IDT para FreeRTOS tem uma ou mais versões correspondentes do FreeRTOS que ele oferece suporte. Recomendamos que você baixe uma nova versão do IDT para FreeRTOS quando uma nova versão do FreeRTOS for lançada.

Ao fazer o download do software, você concorda com o Contrato de Licença do AWS IoT Device Tester contido no arquivo de download.

> **Note**
>
> Ao usar oAWS IoT Device Tester para FreeRTOS, recomendamos que você atualize para a última versão do patch da versão mais recente do FreeRTOS-LTS.

> **Important**
>
> Desde outubro de 2022, o AWS IoT Device Tester para AWS IoT da qualificação do FreeRTOS (FRQ) 1.0 não gera relatórios de qualificação assinados. Você não pode qualificar novos dispositivos AWS IoT do FreeRTOS para serem listados no AWS Partner Device Catalog por meio do Programa de qualificação de dispositivos da AWS usando as versões IDT FRQ 1.0. Embora você não possa qualificar dispositivos FreeRTOS usando o IDT FRQ 1.0, você pode continuar testando seus dispositivos FreeRTOS com o FRQ 1.0. Recomendamos que você use o IDT FRQ 2.0 para qualificar e listar dispositivos FreeRTOS no AWS Partner Device Catalog.

Versão mais recente do AWS IoT Device Tester para FreeRTOS

Use os links a seguir para baixar as versões mais recente do IDT para FreeRTOS.

Versão mais recente do AWS IoT Device Tester para FreeRTOS

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
IDT v4.9.0	FRQ_2.5.0	• 202112.00 • 202212.00 • 202212.01 • Todos os patches do FreeRTOS 202210-LTS que usam bibliotecas do FreeRTOS LTS.	• Linux • macOS • Windows	04/04/2023	• Oferece suporte a testes no FreeRTOS 202112, 202212, 202212.01 e todos os patches 202210-LTS do FreeRTOS que usam bibliotecas do FreeRTOS. Consulte README.md para obter mais informações. Você deve incluir a versão do patch para FreeRTOS-LTS em seu

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
					`manifest.yml` . • Aprimorou o tempo de execução dos testes E2E OTA. • Limita o número de dispositivos listados em `device.json` para 1. • Correções de bugs secundários e melhorias.

> **Note**
>
> Não é recomendável que vários usuários executem o IDT em um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Essa prática pode resultar em falhas ou corrupção de dados. Recomendamos extrair o pacote do IDT para uma unidade local e executar o binário do IDT na estação de trabalho local.

Versões antigas do IDT para o FreeRTOS

As seguintes versões antigas do IDT para FreeRTOS também são compatíveis.

Versões antigas do AWS IoT Device Tester para o FreeRTOS

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
IDT v4.8.1	FRQ_2.4.0	• 202112.00 • 202212.00 • 202212.01 • Todos os patches do FreeRTOS 202210-LTS que usam bibliotecas do FreeRTOS LTS.	• Linux • macOS • Windows	23/01/2023	• Consulte README.MD para obter mais informações. Você deve incluir a versão do patch para FreeRTOS-LTS em seu `manifest.yml` . • Correções de bugs secundários e melhorias.
IDT v4.6.0	FRQ_2.3.0	• 202112.00 • 202212.00 • 202212.01 • 202210-LTS que usam bibliotecas FreeRTOS LTS.	• Linux • macOS • Windows	16/11/2022	• Consulte README.MD para obter mais informações. Você deve incluir a versão do

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
					patch para FreeRTOS-LTS em seu `manifest.yml` . • Para obter mais informações sobre o que está incluído na versão 202210-LTS do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Adiciona a capacidade de configurar e executar o AWS IoT Device Tester para FreeRTOS por meio

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
					de uma interface de usuário baseada na web. Consulte Use a interface de usuário do IDT para FreeRTOS para executar o pacote de qualificação do FreeRTOS 2.0 (FRQ 2.0) para começar. • Adiciona uma opção para reter as cópias modificadas do código-fonte criado e usado no runtime para

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
					depuração pós-teste. Consulte Configuração de parâmetros de compilação, atualização e teste para obter mais informações. • Adiciona suporte ao SDK do cliente IDT para Java. Para obter mais informações sobre o cliente, consulte Usar o IDT para desenvolver e executar os próprios

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
					pacotes de testes.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
IDT v4.5.11	FRQ_2.2.0	• 202112.00 • 202212.00 • 202212.01 • 202210-LTS que usam bibliotecas FreeRTOS LTS.	• Linux • macOS • Windows	14/10/2022	• Consulte README.MD para obter mais informações. Você deve incluir a versão do patch para FreeRTOS-LTS em seu `manifest.yml` . • Para obter mais informações sobre o que está incluído na versão 202210-LTS do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Links para fazer download	Data de lançamento	Notas de release
					• Correções de bugs secundários e melhorias.

Para obter mais informações, consulte Política de suporte do AWS IoT Device Tester para FreeRTOS.

Versões não compatíveis do IDT para FreeRTOS

Esta seção lista as versões não compatíveis do IDT para FreeRTOS. Versões não compatíveis não recebem correções de bugs ou atualizações. Para obter mais informações, consulte Política de suporte do AWS IoT Device Tester para FreeRTOS.

As seguintes versões do IDT-FreeRTOS não são mais compatíveis.

Versões não compatíveis do AWS IoT Device Tester para FreeRTOS

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v4.5.10	FRQ_2.1.4	• 202112.00 • 202012-LTS que usam bibliotecas FreeRTOS LTS.	2/09/2022	• Para obter mais informações sobre o que está incluído na versão 202012-LTS do FreeRTOS, consulte o arquivo

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				CHANGELOG.md no GitHub. • Resolveu um problema que afetava o grupo de teste `OTA End to End`. • Removeu o `FullTransportInterfacePlainText` de ser executado em execuções de qualificação. O texto simples ainda pode ser executado como um grupo de teste de desenvolvimento usando o sinalizador `-\-group-id .` • Melhorou o registro em log, a legibilidade do console e

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				de saída de arquivo. • Correções de bugs secundários e melhorias.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v4.5.9	FRQ_2.1.3	• 202112.00 • 202012.04-LTS que usam bibliotecas FreeRTOS LTS.	17/08/2022	• Para obter mais informações sobre o que está incluído na versão 202012.04-LTS do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Resolveu um problema que afetava o grupo de teste `FreeRTOSIntegrity`. • Atualizou o grupo de teste `FullCloudIoT` removendo o caso de teste "MQTT Connect Exponential Backoff Retries".

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				• Correções de bugs secundários e melhorias.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v4.5.6	FRQ_2.1.2	• 202112.00 • 202012.04-LTS que usam bibliotecas FreeRTOS LTS.	29/06/2022	• Para obter mais informações sobre o que está incluído na versão 202012.04-LTS do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Adiciona um novo grupo de teste `FullCloudIoT` que testa a placa por AWS IoT Core Device Advisor. • Resolveu um problema que afetava os casos de teste OTA E2E. • Correções de bugs

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				secundários e melhorias.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v4.5.5	FRQ_2.1.1	• 202112.00 • 202012.04-LTS que usam bibliotecas FreeRTOS LTS.	06/06/2022	• Para obter mais informações sobre o que está incluído na versão 202012.04-LTS do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Adiciona um novo grupo de teste `FullCloudIoT` que testa a placa por AWS IoT Core Device Advisor. • Resolveu um problema que afetava os casos de teste FreeRTOSVersion e FreeRTOSIntegrity.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				• Correções de bugs secundários e melhorias.
IDT v4.5.5	FRQ_2.1.0	• 202107.00 • 202112.00 • 202012.04-LTS que usam bibliotecas FreeRTOS LTS.	31/05/2022	• Para obter mais informações sobre o que está incluído na versão 202012.04-LTS do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Adiciona um novo grupo de teste `FullCloudIoT` que testa a placa por AWS IoT Core Device Advisor. • Correções de bugs secundários e melhorias.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v4.5.4	FRQ_2.0.0	• 202112.00 • 202012.04-LTS que usam bibliotecas FreeRTOS LTS.	9/05/2022	• Para obter mais informações sobre o que está incluído na versão 202012.04-LTS do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Remove a exigência de qualificar as placas usando somente as versões dos Amazon FreeRTOS do repositório do GitHub `aws/amazon-freertos` . • Correções de bugs secundários e melhorias.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v4.5.2	FRQ_1.6.2	202107.00	25/01/2022	• Para obter mais informações sobre o que está incluído na versão 202107.00 do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Implementa o novo orquestrador de testes do IDT para configurar pacotes de testes personalizados. Para obter mais informações, consulte Configurar o orquestrador de teste do IDT. • Correções de bugs

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				secundários e melhorias.
IDT v4.0.3	FRQ_1.5.1	202012.00	30/07/2021	• Oferece suporte para qualificação de dispositivos com credencias bloqueadas em um módulo de segurança de hardware. • Correções de bugs secundários e melhorias.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v4.3.0	FRQ_1.6.1	202107.00	26/07/2021	• Para obter mais informações sobre o que está incluído na versão 202107.00 do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Adiciona a capacidade de configurar e executar o AWS IoT Device Tester para FreeRTOS por meio de uma interface de usuário baseada na web. Consulte Use a interface de usuário do IDT para FreeRTOS para executar

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				o pacote de qualificação do FreeRTOS para começar.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release

IDT v4.1.0	FRQ_1.6.0	202107.00	21/07/2021	• Para obter mais informações sobre o que está incluído na versão 202107.00 do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub.
				• Remove os seguintes casos de teste da qualificação OTA:
				  • Agente OTA
				  • Nome de arquivo ausente OTA
				  • Número máximo de blocos configurado OTA
				• Remove o grupo de teste `Both`

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				Dataplane OTA da qualificação OTA. No arquivo `device.json`, a configuração `OTADataPlaneProtocol` agora aceita somente `HTTP` ou `MQTT` como valores compatíveis. • Implementa as seguintes alterações na configuração `freertosFileConfiguration` do arquivo `userdata.json` para alterações no código-fonte do FreeRTOS: • Altera o nome do arquivo

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				especific ado para `otaAgentT estsConfi g` e `otaAgentD emosConfi g` de `aws_ota_a gent_conf ig.h` para `ota_confi g.h` . • Adiciona uma configura ção opcional `otaDemosC onfig` nova para especificar o caminho para o arquivo `ota_demo_ config.h` novo. • Adiciona um novo campo `testStart Delayms` para

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				`userdata.json` a fim de especificar um atraso entre o momento em que um dispositivo é instalado para executar um grupo de teste do FreeRTOS e o momento em que ele começa a executar os testes. O valor de deve ser apresentado em milissegundos. Esse atraso pode ser usado para dar ao IDT a chance de se conectar, de forma que nenhuma saída de teste seja perdida.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v4.0.1	FRQ_1.4.1	202012.00	19/01/2021	• Para obter mais informações sobre o que está incluído na versão 202012.00 do FreeRTOS, consulte o arquivo CHANGELOG.md no GitHub. • Apresenta casos de teste E2E (ponta a ponta) OTA (sem fios) adicionais. • Oferece suporte à qualificação de placas de desenvolvimento que executam o FreeRTOS 202012.00 e usam bibliotecas LTS do FreeRTOS.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				• Adiciona suporte para a qualificação de placas de desenvolvimento do FreeRTOS usando conectividade celular. • Corrige um bug na configuração de servidor echo. • Permite que você desenvolva e execute os próprios pacotes de testes personalizados usando o AWS IoT Device Tester para FreeRTOS. Para obter mais informações, consulte Usar

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				o IDT para desenvolver e executar os próprios pacotes de testes. • Fornece aplicativos IDT assinados por código, para que não seja necessári o conceder permissões ao executá-los no Windows ou no macOS. • Refinou a lógica de análise dos resultados do teste de BLE.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v3.4.0	FRQ_1.3.0	202011.01	05/11/2020	• Para obter mais detalhes, consulte o arquivo CHANGELOG.md no GitHub. • Corrigiu o erro em que "RSA" não era uma opção de configuração válida do PKCS11. • Corrigiu o erro em que os buckets do Amazon S3 não eram limpos corretamente após os testes OTA. • Atualizações para oferecer suporte aos novos casos de teste dentro do grupo de teste FullMQTT.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v3.3.0	FRQ_1.2.0	202007.00	17/09/2020	• Para obter mais detalhes, consulte o arquivo CHANGELOG.md no GitHub. • Novos testes de ponta a ponta para validar a suspensão de atualização sem fios e o atributo de retomada. • Corrigido o erro que fazia com que os usuários na Região eu-central-1 não conseguissem passar pela validação de configuração para testes OTA. • Adicionou o parâmetro `--update-idt` ao comando

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				`run-suite` . Você pode usar essa opção para definir a resposta para o prompt de atualização do IDT. • Adicionou o parâmetro `--update-managed-policy` ao comando `run-suite` . Você pode usar essa opção para definir a resposta para o prompt de atualização da política gerenciada. • Melhorias internas e correções de erros, incluindo: • Melhorias na atualizaç

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				ão do arquivo de configura ção para atualizações automáticas de pacote de testes.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v3.0.2	FRQ_1.0.1	202002.00		• Para obter mais detalhes, consulte o arquivo CHANGELOG.md no GitHub. • Adiciona atualização automática de conjuntos de testes no IDT. O IDT agora pode fazer download dos pacotes de testes mais recentes disponíveis para sua versão do FreeRTOS. Com esse recurso, é possível: • Fazer download dos conjuntos de testes mais recentes usando o

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				comando `upgrade-test-suite` . • Fazer download dos conjuntos de testes mais recentes definindo um sinalizador ao iniciar o IDT. Usar a opção `-u` *`flag`* em que *`flag`* pode ser 'y' para sempre fazer download ou 'n' para usar uma versão existente. Quando houver várias versões do conjunto de testes

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				disponíveis, a versão mais recente é usada, a menos que você especifique um ID do conjunto de testes ao iniciar o IDT. • Usar a nova opção `list-supported-versions` para listar as versões do pacote de testes do FreeRTOS compatíveis com a versão instalada do IDT. • Listar casos de teste em um grupo e executar

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				testes individuais. O versionamento dos conjuntos de testes é feito usando um formato `major.minor.patch` começando com 1.0.0. • Adiciona o comando `list-supported-products` – Lista as versões do pacote de testes do FreeRTOS compatíveis com a versão instalada do IDT. • Adiciona o comando `list-test-cases` – Lista os casos de teste que estão

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				disponíveis em um grupo de testes. • Adiciona a opção `test-id` ao comando `run-suite` – Use essa opção para executar casos de teste individuais em um grupo de testes.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v1.7.1	FRQ_1.0.0	202002.00		• Para obter mais detalhes, consulte o arquivo CHANGELOG.md no GitHub. • Oferece suporte ao método de assinatura de código personalizado para casos de teste de ponta a ponta over-the-air (OTA) para que seja possível usar os próprios comandos e scripts de assinatura de código para assinar cargas OTA. • Adiciona uma pré-verificação para portas seriais antes do início dos testes. Os

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				testes falharão rapidamente com mensagens de erro melhoradas se a porta serial estiver mal configurada no arquivo `device.json` . • Adicionou uma Política gerenciada da AWS `AWSIoTDeviceTesterForFreeRTOSFullAccess` com permissões necessárias para executar o AWS IoT Device Tester. Se novas versões exigirem permissões adicionais, as adicionaremos a essa política

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				gerenciada para que não seja necessário atualizar as permissões do IAM manualmente. • O arquivo denominado `AFQ_Report.xml` no diretório de resultados agora é `FRQ_Report.xml`.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v1.6.2	FRQ_1.0.0	202002.00		• Oferece suporte a testes opcionais para OTA via HTTPS para qualificar suas placas de desenvolvimento FreeRTOS. • Oferece suporte ao endpoint ATS de AWS IoT em testes. • Oferece suporte à capacidade de informar os usuários sobre a versão mais recente do IDT antes do início do conjunto de testes.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v1.5.2	FRQ_1.0.0	201910.00		• Oferece suporte à qualificação de dispositivos do FreeRTOS com elemento de segurança (chave integrada). • Oferece suporte a portas de servidor echo configuráveis para grupos de testes de Wi-Fi e Secure Sockets. • Oferece suporte ao sinalizador de multiplic ador de tempo limite para aumentar os tempos limite, o que é útil ao solucionar problemas de erros relaciona dos ao tempo limite.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
				• Adicionada correção de erro para análise de log. • Oferece suporte ao endpoint iot ats em testes.
IDT v1.4.1	FRQ_1.0.0	201908.00		• Adição de suporte para novas atualizações de caso de teste e biblioteca PKCS11. • Códigos de erro acionáveis introduzidos. Para obter mais informações, consulte Códigos de erro de IDT. • Atualização da política do IAM usada para executar o IDT.

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT v1.3.2	FRQ_1.0.0	201906.00		• Adicionado suporte para testes de Bluetooth Low Energy (BLE). • Aprimoramento da experiência do usuário para comandos de interface de linha de comando (CLI) do IDT. • Atualização da política do IAM usada para executar o IDT.
IDT-FreeRTOS v1.2	FRQ_1.0.0	• FreeRTOS v1.4.8 • FreeRTOS v1.4.9		Adicionou suporte para testes de dispositivos FreeRTOS com o sistema de compilação CMAKE.
IDT-FreeRTOS v1.1	FRQ_1.0.0			

Versão do AWS IoT Device Tester	Versões do pacote de testes	Versões compatíveis do FreeRTOS	Data de lançamento	Notas de release
IDT-FreeRTOS v1.0	FRQ_1.0.0			

Faça download do IDT para FreeRTOS

Este tópico descreve as opções de download do IDT para FreeRTOS. Você pode usar um dos links de download de software a seguir ou seguir as instruções para baixar o IDT de forma programada.

Important

Desde outubro de 2022, o AWS IoT Device Tester para AWS IoT da qualificação do FreeRTOS (FRQ) 1.0 não gera relatórios de qualificação assinados. Você não pode qualificar novos dispositivos AWS IoT do FreeRTOS para serem listados no AWS Partner Device Catalog por meio do Programa de qualificação de dispositivos da AWS usando as versões IDT FRQ 1.0. Embora você não possa qualificar dispositivos FreeRTOS usando o IDT FRQ 1.0, você pode continuar testando seus dispositivos FreeRTOS com o FRQ 1.0. Recomendamos que você use o IDT FRQ 2.0 para qualificar e listar dispositivos FreeRTOS no AWS Partner Device Catalog.

Tópicos

- Baixar IDT manualmente
- Baixar IDT de maneira programada

Ao fazer o download do software, você concorda com o Contrato de Licença do AWS IoT Device Tester contido no arquivo de download.

Note

O IDT não oferece suporte a execução por vários usuários em um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Recomendamos extrair

o pacote do IDT para uma unidade local e executar o binário do IDT na estação de trabalho local.

Baixar IDT manualmente

Este tópico lista as versões compatíveis do IDT para FreeRTOS. Como prática recomendada, aconselhamos usar a versão mais recente do IDT para AWS IoT Device Tester que seja compatível com a sua versão de destino do FreeRTOS. Novas versões do FreeRTOS podem exigir que você faça o download de uma nova versão do IDT para AWS IoT Device Tester. Você receberá uma notificação ao iniciar uma execução de teste se o IDT para AWS IoT Device Tester não for compatível com a versão do FreeRTOS que você está usando.

Consulte Versões compatíveis do AWS IoT Device Tester para FreeRTOS

Baixar IDT de maneira programada

O IDT fornece uma operação de API que você pode usar para recuperar um URL na qual é possível baixar o IDT de maneira programada. Você também pode usar essa operação de API para verificar se você tem a versão mais recente do IDT. Essa operação da API tem o seguinte endpoint.

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

Para chamar essa operação de API, você deve ter a permissão para executar a ação **`iot-device-tester:LatestIdt`**. Inclua sua assinatura da AWS, com `iot-device-tester` como o nome do serviço

Solicitações de API

HostOS: o sistema operacional da máquina host. Escolha entre as seguintes opções:

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

TestSuiteType: o tipo do pacote de testes. Escolha a seguinte opção:

`FR`: IDT para FreeRTOS

ProductVersion

(Opcional) A versão do FreeRTOS. O serviço retorna a versão mais recente compatível do IDT para essa versão do FreeRTOS. Se você não especificar essa opção, o serviço retornará a versão mais recente do IDT.

Resposta da API

O resposta da API tem o seguinte formato. O arquivo `DownloadURL` inclui o arquivo 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
    }
 }
```

Exemplos

Você pode consultar os exemplos a seguir para baixar o IDT de maneira programada. Esses exemplos usam credenciais que você armazena nas variáveis de ambiente `AWS_ACCESS_KEY_ID` e `AWS_SECRET_ACCESS_KEY`. Para seguir as práticas recomendadas de segurança, não armazene suas credenciais em seu código.

Example

Exemplo: faça download usando cURL versão 7.75.0 ou posterior (Mac e Linux)

Se você tiver a versão 7.75.0 ou posterior do cURL, poderá usar a sinalização `aws-sigv4` para assinar a solicitação da API. Este exemplo usa jq para analisar o URL de download da resposta.

> **Warning**
>
> A sinalização `aws-sigv4` exige que os parâmetros de consulta da solicitação GET curl estejam na ordem HostOs/ProductVersion/TestSuiteType ou HostOs/TestSuiteType. Pedidos que não estiverem em conformidade resultarão em um erro ao obter assinaturas incompatíveis para a string canônica do API Gateway.

Se o parâmetro opcional ProductVersion estiver incluído, você deverá usar uma versão de produto compatível conforme documentado em Versões compatíveis do AWS IoT Device Tester para FreeRTOS.

- Substitua *us-west-2* pela sua Região da AWS. Para obter uma lista de códigos de região, consulte Endpoints regionais.
- Substitua *linux* pelo sistema operacional da sua máquina host.
- Substitua *202107.00* pela sua versão dos 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

Exemplo: faça download usando uma versão anterior do cURL (Mac e Linux)

Você pode usar o seguinte comando cURL com uma assinatura da AWS que você assina e calcula. Para obter mais informações sobre como assinar e calcular uma assinatura da AWS, consulte Assinatura de solicitações de API da AWS.

- Substitua *linux* pelo sistema operacional da sua máquina host.
- Substitua *Timestamp* pela data e hora, como **20220210T004606Z**.
- Substitua a *date* pela data, como **20220210**.
- Substitua a *AWSRegion* pela sua Região da AWS. Para obter uma lista de códigos de região, consulte Endpoints regionais.
- Substitua *AWSSignature* pela Assinatura da AWS gerada.

```
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

Exemplo: faça o download usando um script Python

Este exemplo usa a biblioteca de solicitações do Python. Esse exemplo foi adaptado do exemplo do Python para Assinar uma solicitação de API da AWS na Referência geral da AWS.

- Substitua *us-west-2* pela sua região. Para obter uma lista de códigos de região, consulte Endpoints regionais.
- Substitua *linux* pelo sistema operacional da sua 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)
```

Use o IDT com o pacote de qualificação do FreeRTOS 2.0 (FRQ 2.0)

O pacote de qualificação do FreeRTOS 2.0 é uma versão atualizada do pacote de qualificação do FreeRTOS. É recomendado que os desenvolvedores usem o FRQ 2.0 porque ele consiste em casos de teste relevantes para qualificar dispositivos que executam as bibliotecas do FreeRTOS Long Term Support (LTS).

O IDT para FreeRTOS verifica a porta do FreeRTOS em seu microcontrolador e se ela pode se comunicar de maneira efetiva com o AWS IoT. Especificamente, ele verifica as interfaces da camada de portabilidade com as bibliotecas do FreeRTOS e se os repositórios de teste do FreeRTOS estão implementados corretamente. Ele também executa testes de ponta a ponta com o AWS IoT Core. Os testes executados pelo IDT para FreeRTOS são definidos no Repositório de GitHub do FreeRTOS.

O IDT para FreeRTOS executa testes como aplicações incorporadas que são exibidas no dispositivo microcontrolador em teste. As imagens binárias da aplicação incluem o FreeRTOS, as interfaces do FreeRTOS obtidas por portabilidade e os drivers de dispositivos da placa. A finalidade dos testes é verificar se as interfaces do FreeRTOS obtidas por portabilidade funcionam corretamente em cima dos drivers do seu dispositivo.

O IDT para FreeRTOS gera relatórios de teste que você pode enviar para o AWS IoT para ter seu hardware listado no AWS Partner Device Catalog. Para obter mais informações, consulte AWS Device Qualification Program.

O IDT para FreeRTOS é executado em um computador host (Windows, macOS ou Linux) que está conectado à placa em teste. O IDT configura e orquestra casos de teste e reúne resultados. Ele também fornece uma interface de linha de comando para gerenciar a execução dos testes.

Para testar seu dispositivo, o IDT for FreeRTOS cria recursos como coisas de AWS IoT, grupos do FreeRTOS e funções do Lambda. Para criar esses recursos, o IDT para FreeRTOS usa as credenciais da AWS configuradas no `config.json` para fazer chamadas de API em seu nome. Esses recursos são provisionados em vários momentos durante o teste.

Ao executar o IDT para FreeRTOS em seu computador host, ele executa as seguintes etapas:

1. Carrega e valida o dispositivo e configuração de credenciais.
2. Executa testes selecionados com os recursos locais e de nuvem necessários.
3. Remove recursos locais e de nuvem.
4. Gera relatórios de testes que indicam se a placa passou nos testes necessários para a qualificação.

Tópicos

Pré-requisitos

Esta seção descreve os pré-requisitos para testar microcontroladores com o AWS IoT Device Tester.

Preparar-se para a qualificação do FreeRTOS

> **Note**
>
> AWS IoT Device Tester para FreeRTOS recomenda o uso da versão de patch mais recente da versão mais recente do FreeRTOS-LTS.

O IDT para FRQ 2.0 é uma qualificação para FreeRTOS. Antes de executar o IDT FRQ 2.0 para qualificação, é preciso concluir a Qualificação de sua placa no Guia de qualificação do FreeRTOS. Para portar bibliotecas, testar e configurar o `manifest.yml`, consulte Como portar as bibliotecas

do FreeRTOS no Guia de portabilidade do FreeRTOS. O FRQ 2.0 contém um processo diferente de qualificação. Consulte Alterações mais recentes na qualificação no Guia de qualificação do FreeRTOS para obter detalhes.

O repositório IDT-FreeRTOS-Libraries-Integration-Tests deve estar presente para que o IDT seja executado. Consulte README.md sobre como clonar e portar esse repositório para seu projeto de origem. Os FreeRTOS-Libraries-Integration-Tests devem incluir o `manifest.yml` localizado na raiz do seu projeto para que o IDT seja executado.

> **Note**
>
> O IDT depende da implementação do repositório de testes do `UNITY_OUTPUT_CHAR`. Os logs de saída de teste e os logs do dispositivo não devem ser intercalados entre si. Consulte Como implementar a seção de macros de registro em log da biblioteca no Guia de portabilidade do FreeRTOS para obter mais detalhes.

Baixe o IDT para FreeRTOS

Cada versão do FreeRTOS tem uma versão correspondente do IDT para o FreeRTOS para realizar testes de qualificação. Baixe a versão adequada do IDT para FreeRTOS das versões compatíveis do AWS IoT Device Tester para FreeRTO..

Extraia o IDT para FreeRTOS em um local no sistema de arquivos onde existam permissões de leitura e gravação. Como o Microsoft Windows tem um limite de caracteres para o comprimento do caminho, extraia o IDT para FreeRTOS em um diretório raiz, como `C:\` ou `D:\`.

> **Note**
>
> Vários usuários não devem executar o IDT de um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Isto pode resultar em falhas ou corrupção de dados. Recomendamos extrair o pacote IDT para uma unidade local.

Baixe o Git

O IDT deve ter o Git instalado como um pré-requisito para garantir a integridade do código-fonte.

Siga as instruções no guia do GitHub para instalar o Git. Para verificar a versão atual instalada do Git, digite o comando `git --version` no terminal.

> **Warning**
>
> O IDT usa o Git para se alinhar ao status de limpo ou sujo de um diretório. Se o Git não estiver instalado, os grupos de teste `FreeRTOSIntegrity` falharão ou não serão executados conforme o esperado. Se o IDT retornar um erro como `git executable not found` ou `git command not found`, instale ou reinstale o Git e tente novamente.

Criação e configuração de uma conta da AWS

> **Note**
>
> O pacote completo de qualificação da IDT é compatível somente nas seguintes Regiões da AWS
>
> - Leste dos EUA (N. da Virgínia)
> - Oeste dos EUA (Oregon)
> - Ásia-Pacífico (Tóquio)
> - Europa (Irlanda)

Para testar seu dispositivo, o IDT for FreeRTOS cria recursos como coisas de AWS IoT, grupos do FreeRTOS e funções do Lambda. Para criar esses recursos, o IDT for FreeRTOS exige que você crie e configure uma conta da AWS e uma política do IAM que conceda ao IDT for FreeRTOS permissão para acessar recursos em seu nome durante a execução de testes.

Use as etapas a seguir para criar e configurar sua conta da AWS.

1. Se já tiver uma conta da AWS, passe para a próxima etapa. Para criar uma conta da AWS.
2. Siga as etapas em Como criar perfis do IAM. Não adicione permissões ou políticas no momento.
3. Para executar testes de qualificação OTA, vá para a etapa 4. Caso contrário, vá para a etapa 5.
4. Anexe a política embutida de permissões do OTA IAM ao seu perfil do IAM.

   a.

   > **Important**
   >
   > O modelo de política a seguir concede ao IDT permissão para criar funções e políticas, e associar políticas às funções. O IDT para FreeRTOS usa essas

permissões para testes que criam perfis. Embora o modelo de política não forneça privilégios de administrador ao usuário, as permissões podem ser usadas para obter acesso de administrador à conta da AWS.

b. Siga as etapas abaixo para anexar as permissões necessárias ao perfil do IAM:

i. Na guia Permissões, escolha Adicionar permissões.

ii. Escolha Criar política em linha.

iii. Escolha a guia JSON e copie as permissões a seguir na caixa de texto JSON. Use o modelo na Maioria das regiões se não estiver na região da China. Se estiver na região da China, use o modelo nas regiões de Pequim e 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

O modelo de política a seguir pode ser usado nas regiões de Pequim e 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. Ao concluir, selecione Review policy (Revisar política).

v. Insira IDTFreeRTOSIAMPermissions como o nome da política.

vi. Escolha Create policy (Criar política).

5. Anexe AWSIoTDeviceTesterForFreeRTOSFullAccess ao seu perfil do IAM.

   a. Para associar as permissões necessárias ao seu perfil do IAM:

      i. Na guia Permissões, escolha Adicionar permissões.

      ii. Escolha Attach policies (Anexar políticas).

      iii. Procure a política AWSIoTDeviceTesterForFreeRTOSFullAccess. Marque a caixa.

   b. Escolha Add permissions (Adicionar permissões).

6. Exporte credenciais para o IDT. Consulte Obter credenciais de perfil do IAM para acesso à CLI para obter detalhes.

Política gerenciada pelo AWS IoT Device Tester

A política gerenciada `AWSIoTDeviceTesterForFreeRTOSFullAccess` contém as seguintes permissões do AWS IoT Device Tester para a verificação de versão, atributos de atualização automática e coleção de métricas.

- `iot-device-tester:SupportedVersion`

  Concede permissão ao AWS IoT Device Tester para buscar a lista de produtos compatíveis, pacotes de teste e versões do IDT.

- `iot-device-tester:LatestIdt`

Concede permissão ao AWS IoT Device Tester para obter a versão mais recente do IDT disponível para download.

- `iot-device-tester:CheckVersion`

  Concede permissão ao AWS IoT Device Tester para verificar a compatibilidade de versões do IDT, pacotes de teste e produtos.

- `iot-device-tester:DownloadTestSuite`

  Concede permissão ao AWS IoT Device Tester para fazer download de pacotes de teste.

- `iot-device-tester:SendMetrics`

  Concede permissão à AWS para coletar métricas sobre o uso interno do AWS IoT Device Tester.

(Opcional) Instalar a AWS Command Line Interface

Talvez você prefira usar a AWS CLI para executar algumas operações. Se não tiver a AWS CLI instalada, siga as instruções em Instalar a AWS CLI.

Configure a AWS CLI para a região da AWS desejada, executando aws configure em uma linha de comando. Para obter informações sobre as regiões da AWS compatíveis com o IDT para FreeRTOS, consulte Regiões e endpoints da AWS. Para obter mais informações sobre aws configure, consulte Configuração rápida com o aws configure.

Preparação para testar sua placa de microcontrolador pela primeira vez

É possível usar o IDT for FreeRTOS para testar sua implementação das bibliotecas do FreeRTOS. Depois de transferir as bibliotecas do FreeRTOS para os drivers de dispositivo da placa, use o AWS IoT Device Tester para executar testes de qualificação na placa do microcontrolador.

Adicione camadas de portabilidade da biblioteca e implemente um repositório de teste do FreeRTOS

Para fazer a portabilidade do FreeRTOS para seu dispositivo, consulte o Guia de portabilidade do FreeRTOS. Ao implementar o repositório de testes do FreeRTOS e portar as camadas do FreeRTOS, é preciso fornecer um `manifest.yml` com caminhos para cada biblioteca, incluindo o repositório de testes. Este arquivo estará no diretório-raiz do código-fonte. Consulte as instruções do arquivo de manifesto para obter detalhes.

Configurar as credenciais da AWS

É preciso configurar suas credenciais da AWS para que o AWS IoT Device Tester se comunique com a nuvem da AWS. Para obter mais informações, consulte Configurar as credenciais e a região da AWS para desenvolvimento. As credenciais válidas da AWS são especificadas no arquivo de configuração *devicetester_extract_location*/devicetester_freertos_*[win|mac|linux]*/configs/config.json.

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

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

O atributo `auth` do arquivo `config.json` tem um campo de método que controla a autenticação da AWS e pode ser declarado como arquivo ou ambiente. Definir o campo como ambiente extrai suas credenciais da AWS das variáveis de ambiente da sua máquina host. Definir o campo como arquivo importa um perfil especificado do arquivo de configuração `.aws/credentials`.

Crie um grupo de dispositivos no IDT para FreeRTOS

Os dispositivos a serem testados são organizados em grupos de dispositivos. Cada grupo de dispositivos consiste em um ou mais dispositivos idênticos. Também é possível configurar o IDT para FreeRTOS para testar um único dispositivo ou vários dispositivos em um grupo. Para acelerar o processo de qualificação, o IDT para FreeRTOS pode testar dispositivos com as mesmas especificações em paralelo. Ele usa o método round-robin para executar um grupo de testes diferente em cada dispositivo de um grupo de dispositivos.

O arquivo `device.json` tem uma matriz em seu nível superior. Cada atributo da matriz é um novo grupo de dispositivos. Cada grupo de dispositivos tem um atributo de matriz de dispositivos, que tem vários dispositivos declarados. No modelo, há um grupo de dispositivos e somente um dispositivo neste grupo de dispositivos. Você pode adicionar um ou mais dispositivos a um grupo de dispositivos editando a seção `devices` do modelo `device.json` na pasta `configs`.

> **Note**
>
> Todos os dispositivos no mesmo grupo devem ser da mesma especificação técnica e SKU. Para habilitar as compilações paralelas do código-fonte para diferentes grupos de teste, o IDT para FreeRTOS copia o código-fonte para uma pasta de resultados dentro da pasta onde o IDT para FreeRTOS foi extraído. É preciso referenciar o caminho do código-fonte no comando de compilação ou atualização por meio da variável `testdata.sourcePath`. O IDT para FreeRTOS substitui esta variável por um caminho temporário do código-fonte copiado. Para obter mais informações, consulte Variáveis do IDT para FreeRTOS.

Veja a seguir um exemplo do arquivo `device.json` que foi usado para criar um grupo de dispositivos com vários 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"
              }
            ]
          }
        ]
    }
]
```

Os seguintes atributos são usados no arquivo `device.json`:

`id`

Um ID alfanumérico definido pelo usuário que identifica exclusivamente um grupo de dispositivos. Os dispositivos que pertencem a um grupo devem ser do mesmo tipo. Quando um conjunto

de testes está em execução, os dispositivos do grupo são usados para paralelizar a carga de trabalho.

`sku`

Um valor alfanumérico que identifica exclusivamente a placa sendo testada. A SKU é usada para rastrear as placas qualificadas.

> **Note**
>
> Se deseja listar sua placa no AWS Partner Device Catalog, a SKU especificada aqui deve corresponder à SKU usada no processo de oferta.

`features`

Uma matriz contendo os atributos compatíveis do dispositivo. O AWS IoT Device Tester usa essas informações para selecionar os testes de qualificação a serem executados.

Os valores compatíveis são:

`Wifi`

Indica se a placa tem recursos de Wi-Fi.

`Cellular`

Indica se a placa tem recursos de celular.

`PKCS11`

Indica o algoritmo de criptografia de chave pública compatível com a placa. O PKCS11 é necessário para qualificação. Os valores compatíveis são `ECC`, `RSA`, e `Both`. `Both` indica que a placa é compatível com os algoritmos `ECC` e `RSA`.

`KeyProvisioning`

Indica o método de gravação de um certificado de cliente X.509 confiável em sua placa.

Os valores válidos são `Import`, `Onboard`, `Both` e `No`. O provisionamento das chaves `Onboard`, `Both` ou `No` é necessário para a qualificação. `Import`, por si só, não é uma opção válida para qualificação.

- Use `Import` somente se a sua placa permitir a importação de chaves privadas. Selecionar `Import` não é uma configuração válida para qualificação e deve ser usada somente para

fins de teste, especificamente com casos de teste do PKCS11. `Onboard`, `Both` ou `No` é necessário para a qualificação.

- Use `Onboard` se a placa for compatível com chaves privadas integradas (por exemplo, se o dispositivo tiver um elemento seguro ou se você preferir gerar o seu próprio par de chaves de dispositivo e certificado). Adicione um elemento `secureElementConfig` em cada uma das seções do dispositivo e coloque o caminho absoluto para o arquivo de chave pública no campo `publicKeyAsciiHexFilePath`.
- Use `Both` se sua placa suportar a importação de chaves privadas e a geração de chaves integradas para provisionamento de chaves.
- Use `No` se sua placa não suportar o provisionamento de chaves. `No` só é uma opção válida quando seu dispositivo também está pré-provisionado.

`OTA`

Indica se a placa oferece suporte à funcionalidade de atualização remota (OTA — Over-the-air). O atributo `OtaDataPlaneProtocol` indica a qual protocolo de plano de dados OTA o dispositivo oferece suporte. Um OTA com o protocolo de plano de dados HTTP ou MQTT é necessário para a qualificação. Para pular a execução de testes OTA durante o teste, defina o recurso OTA como `No` e o atributo `OtaDataPlaneProtocol` como `None`. Esta não será uma corrida de qualificação.

`BLE`

Indica se a placa é compatível com Bluetooth Low Energy (BLE).

`devices.id`

Um identificador exclusivo, definido pelo usuário, para o dispositivo que está sendo testado.

`devices.connectivity.serialPort`

A porta serial do computador host usada para se conectar aos dispositivos que estão sendo testados.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Obrigatório se sua placa NÃO for `pre-provisioned` ou `PublicDeviceCertificateArn` não for fornecido. Como `Onboard` é um tipo obrigatório de provisionamento de chaves, este campo é obrigatório atualmente para o grupo de teste FullTransportInterfaceTLS. Se o seu dispositivo for `pre-provisioned`, `PublicKeyAsciiHexFilePath` é opcional e não precisa ser incluído.

O bloco a seguir é um caminho absoluto para o arquivo que contém a chave pública de bytes hexadecimal extraída da chave 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
```

Se sua chave pública estiver no formato .der, é possível codificar a chave pública em hexadecimal diretamente para gerar o arquivo hexadecimal.

Para gerar o arquivo hexadecimal a partir de uma chave pública .der, digite o seguinte comando xxd:

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

Se sua chave pública estiver no formato .pem, é possível extrair os cabeçalhos e rodapés codificados em base64 e decodificá-los em formato binário. Em seguida, string binária é codificada em hexadecimal para gerar o arquivo hexadecimal.

Para gerar um arquivo hexadecimal para uma chave pública. pem, faça o seguinte:

1. Execute o comando base64 a seguir para remover o cabeçalho e o rodapé base64 da chave pública. A chave decodificada, chamada `base64key`, é enviada para o arquivo `pubkey.der`:

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

2. Execute o comando xxd a seguir para converter `pubkey.der` para o formato hexadecimal. A chave resultante é salva como *`outFile`*.

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

`devices.secureElementConfig.PublicDeviceCertificateArn`

O ARN do certificado do seu elemento seguro que é carregado para AWS IoT Core. Para obter informações sobre como fazer upload do seu certificados para AWS IoT Core, consulte Certificados de cliente X.509 no Guia do desenvolvedor do AWS IoT.

`devices.secureElementConfig.SecureElementSerialNumber`

(Opcional) O número de série do elemento seguro. O número de série é usado como opção para criar certificados de dispositivo para o provisionamento de chaves JITR.

`devices.secureElementConfig.preProvisioned`

(Opcional) Defina como "Sim" se o dispositivo tiver um elemento seguro pré-provisionado com credenciais bloqueadas, que não possa importar, criar ou destruir objetos. Se este atributo for definido como Sim, os rótulos pkcs11 correspondentes deverão ser fornecidos.

`devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport`

(Opcional) Defina como Sim se a implementação do CorePKCS11 do dispositivo for compatível com o armazenamento para JITP. Isto habilitará o teste `codeverify` JITP ao testar o PKCS 11 principal e exigirá que sejam fornecidos rótulos de chave de verificação de código, certificado JITP e certificado raiz PKCS 11.

`identifiers`

(Opcional) Uma matriz de pares de nome-valor arbitrários. Você pode usar esses valores nos comandos de compilação e atualização descritos na próxima seção.

Configuração de parâmetros de compilação, atualização e teste

O IDT para FreeRTOS compila e atualiza os testes automaticamente em sua placa. Para habilitar isso, é preciso configurar o IDT para executar os comandos compilar e atualizar do seu hardware. As configurações de comando de compilação e atualização são definidas no modelo de arquivo `userdata.json` localizado na pasta `config`.

Configuração de parâmetros para testar dispositivos

As configurações de compilação, atualização e teste são feitas no arquivo `configs/userdata.json`. O seguinte exemplo de JSON mostra como é possível configurar o IDT para FreeRTOS para testar vários 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>"
     }
  }
```

Veja a seguir uma lista dos atributos usados no `userdata.json`:

`sourcePath`

O caminho para a raiz do código-fonte transferido do FreeRTOS.

`retainModifiedSourceDirectories`

(Opcional) Verifica se deve reter os diretórios de origem modificados usados durante a compilação e a atualização para fins de depuração. Se definido como `true`, os diretórios de origem modificados são denominados RetainedSrc e são encontrados nas pastas de log de

resultados em cada execução do grupo de teste. Se não for incluído, o campo assume `false` como padrão.

`freeRTOSTestParamConfigPath`

O caminho para o arquivo `test_param_config.h` para a integração FreeRTOS-Libraries-Integration-Tests integration. Este arquivo deve usar a variável de espaço reservado `{{testData.sourcePath}}` para torná-lo relativo à raiz do código-fonte. O AWS IoT Device Tester usa os parâmetros neste arquivo para configurar os testes.

`freeRTOSTestExecutionConfigPath`

O caminho para o arquivo `test_execution_config.h` para a integração FreeRTOS-Libraries-Integration-Tests integration. Este arquivo deve usar a variável de espaço reservado `{{testData.sourcePath}}` para torná-lo relativo à raiz do repositório. O AWS IoT Device Tester usa este arquivo para controlar quais testes devem ser executados.

`freeRTOSVersion`

A versão dos FreeRTOS, incluindo a versão do patch usada em sua implementação. Consulte Versões compatíveis com o AWS IoT Device Tester para FreeRTOS para as versões do FreeRTOS compatíveis com o AWS IoT Device Tester FreeRTOS.

`buildTool`

O comando para compilar o código-fonte. Todas as referências ao caminho do código-fonte no comando de compilação devem ser substituídas pela variável `{{testData.sourcePath}}` do AWS IoT Device Tester. Use o espaço reservado `{{config.idtRootPath}}` para referenciar um script de construção em relação ao caminho-raiz AWS IoT Device Tester.

`flashTool`

O comando para exibir uma imagem em seu dispositivo. Todas as referências ao caminho do código-fonte no comando de atualização devem ser substituídas pela variável `{{testData.sourcePath}}` do AWS IoT Device Tester. Use o espaço reservado `{{config.idtRootPath}}` para referenciar um script de atualização em relação ao caminho-raiz AWS IoT Device Tester.

> **Note**
>
> A nova estrutura de testes de integração com o FRQ 2.0 não requer variáveis de caminho como `{{enableTests}}` e `{{buildImageName}}`. Os testes do OTA End to End

são executados com os modelos de configuração fornecidos no repositório FreeRTOS-Libraries-Integration-Tests integration. do GitHub. Se os arquivos no repositório do GitHub estiverem presentes no seu projeto fonte principal, o código-fonte não será alterado entre os testes. Se for necessária uma imagem de compilação diferente para o OTA End to End, esta imagem deverá ser criada no script de compilação e especificada no arquivo `userdata.json` especificado abaixo `otaConfiguration`.

`testStartDelayms`

Especifica quantos milissegundos o executor de testes do FreeRTOS aguardará antes de começar a executar os testes. Isto pode ser útil se o dispositivo em teste começar a gerar informações de teste importantes antes que o IDT tenha a chance de se conectar e iniciar o registro em log devido à rede ou outros problemas de latência. Este valor é aplicável somente aos grupos de teste do FreeRTOS, e não a outros grupos de teste que não utilizam o executor de testes do FreeRTOS, como os testes OTA. Se receber um erro relacionado a 10 esperados, mas cinco recebidos, este campo deverá ser definido como 5.000.

`echoServerConfiguration`

A configuração para configurar o servidor de eco para o teste TLS. Este campo é obrigatório.

`keyGenerationMethod`

O servidor echo está configurado com esta opção. As opções são EC ou RSA.

`serverPort`

O número da porta na qual o servidor echo é executado.

`otaConfiguration`

A configuração para testes OTA PAL e OTA E2E. Este campo é obrigatório.

`otaE2EFirmwarePath`

O caminho para a imagem do compartimento OTA que o IDT usa para os testes do OTA End to End.

`otaPALCertificatePath`

O caminho para o certificado para o teste OTA PAL no dispositivo. Isto é usado para verificar a assinatura. Por exemplo, ecdsa-sha256-signer.crt.pem.

`deviceFirmwarePath`

O caminho para o nome com codificação rígida da imagem do firmware a ser inicializada. Se o seu dispositivo NÃO usar o sistema de arquivos para inicialização do firmware, especifique este campo como `'NA'`. Se o seu dispositivo usa o sistema de arquivos para inicialização do firmware, especifique o caminho ou o nome da imagem de inicialização do firmware.

`codeSigningConfiguration`

`signingMethod`

O método de assinatura de código. Os valores possíveis são da AWS ou Personalizado.

> **Note**
>
> Nas regiões de Pequim e Ningxia, use Personalizado. Não há compatibilidade para a assinatura de código da AWS nessa região.

`signerHashingAlgorithm`

O algoritmo de hash ao qual o dispositivo oferece suporte. Os valores possíveis são `SHA1` ou `SHA256`.

`signerSigningAlgorithm`

O algoritmo de assinatura ao qual o dispositivo oferece suporte. Os valores possíveis são `RSA` ou `ECDSA`.

`signerCertificate`

O certificado de confiança usado para OTA. Para o método de assinatura de código da AWS, use o nome do recurso da Amazon (ARN) para o certificado confiável carregado para o AWS Certificate Manager. Para o método de assinatura de código personalizado, use o caminho absoluto para o arquivo de certificado do assinante. Para obter mais informações sobre como criar um certificado confiável, consulte Criar um certificado de assinatura de código.

`untrustedSignerCertificate`

O ARN ou caminho de arquivo de um segundo certificado usado em alguns testes OTA como um certificado não confiável. Para obter informações sobre como criar um certificado, consulte Criar um certificado de assinatura de código.

`signerCertificateFileName`

O nome do arquivo do certificado de assinatura de código no dispositivo. Este valor deve corresponder ao nome do arquivo fornecido ao executar o comando `aws acm import-certificate`.

`compileSignerCertificate`

Valor booleano que determina o status do certificado de verificação de assinatura. Os valores válidos são `true` e `false`.

Defina este valor como verdadeiro se o certificado de verificação da assinatura do signatário do código não for provisionado ou exibido. Ele deve ser compilado no projeto. AWS IoT Device Tester busca o certificado confiável e o compila em `aws_codesigner_certificate.h`.

`signerPlatform`

O algoritmo de assinatura e hash que o AWS Código Signer usa ao criar o trabalho de atualização OTA. Atualmente, os valores possíveis para este campo são `AmazonFreeRTOS-TI-CC3220SF` e `AmazonFreeRTOS-Default`.

- Escolha `AmazonFreeRTOS-TI-CC3220SF` se `SHA1` e `RSA`.
- Escolha `AmazonFreeRTOS-Default` se `SHA256` e `ECDSA`.
- Se você precisar de `SHA256` | `RSA` ou `SHA1` | `ECDSA` para sua configuração, entre em contato conosco para obter mais suporte.
- Configure `signCommand` se você escolheu `Custom` para `signingMethod`.

`signCommand`

Dois espaços reservados `{{inputImageFilePath}}` e `{{outputSignatureFilePath}}` são necessários no comando. `{{inputImageFilePath}}` é o caminho do arquivo da imagem criada pelo IDT a ser assinado. `{{outputSignatureFilePath}}` é o caminho do arquivo da assinatura que será gerado pelo script.

`pkcs11LabelConfiguration`

A configuração do rótulo PKCS11 requer pelo menos um conjunto de rótulos de rótulo de certificado de dispositivo, rótulo de chave pública e rótulo de chave privada para executar os grupos de teste do PKCS11. Os rótulos PKCS11 necessários são baseadas na configuração do dispositivo no arquivo `device.json`. Se pré-provisionado estiver definido como Sim em

`device.json`, os rótulos necessários deverão ser um dos abaixo, dependendo do que for escolhido para o atributo PKCS11.

- `PreProvisionedEC`
- `PreProvisionedRSA`

Se pré-provisionado estiver definido como Não em `device.json`, os rótulos necessários serão:

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

Os três rótulos a seguir são necessários somente se você selecionar Sim para `pkcs11JITPCodeVerifyRootCertSupport` em seu arquivo `device.json`.

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

Os valores desses campos devem corresponder aos valores definidos no Guia de portabilidade do FreeRTOS.

`pkcs11LabelDevicePrivateKeyForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 da chave privada. Para dispositivos com suporte integrado e de importação de provisionamento de chaves, este rótulo é usado para testes. Este rótulo pode ser diferente daquele definido para o caso pré-provisionado. Se tiver o provisionamento de chaves definido como Não e o pré-provisionado definido como Sim, em `device.json`, isso será indefinido.

`pkcs11LabelDevicePublicKeyForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 da chave pública. Para dispositivos com suporte integrado e de importação de provisionamento de chaves, este rótulo é usado para testes. Este rótulo pode ser diferente do definido para o caso pré-provisionado. Se tiver o provisionamento de chaves definido como Não e o pré-provisionado definido como Sim, em `device.json`, isso será indefinido.

`pkcs11LabelDeviceCertificateForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 do certificado do dispositivo. Para dispositivos com suporte integrado e de importação de provisionamento de chaves, este

rótulo será usado para testes. Este rótulo pode ser diferente do definido para o caso pré-provisionado. Se tiver o provisionamento de chaves definido como Não e o pré-provisionado definido como Sim, em `device.json`, isso será indefinido.

`pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 da chave privada. Para dispositivos com elementos seguros ou limitações de hardware, isto terá um rótulo diferente para preservar as credenciais do AWS IoT. Se o seu dispositivo for compatível com o pré-provisionamento com uma chave EC, forneça este rótulo. Quando PreProvisioned é definido como Sim em `device.json`, este rótulo, `pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS`, ou ambos devem ser fornecidos. Este rótulo pode ser diferente do definido para caixas integradas e de importação.

`pkcs11LabelPreProvisionedECDevicePublicKeyForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 da chave pública. Para dispositivos com elementos seguros ou limitações de hardware, isto terá um rótulo diferente para preservar as credenciais do AWS IoT. Se o seu dispositivo for compatível com o pré-provisionamento com uma chave EC, forneça este rótulo. Quando PreProvisioned é definido como Sim em `device.json`, este rótulo, `pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS`, ou ambos devem ser fornecidos. Este rótulo pode ser diferente do definido para caixas integradas e de importação.

`pkcs11LabelPreProvisionedECDeviceCertificateForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 do certificado do dispositivo. Para dispositivos com elementos seguros ou limitações de hardware, isto terá um rótulo diferente para preservar as credenciais do AWS IoT. Se o seu dispositivo for compatível com o pré-provisionamento com uma chave EC, forneça este rótulo. Quando PreProvisioned é definido como Sim em `device.json`, este rótulo, `pkcs11LabelPreProvisionedRSADeviceCertificateForTLS`, ou ambos devem ser fornecidos. Este rótulo pode ser diferente do definido para caixas integradas e de importação.

`pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 da chave privada. Para dispositivos com elementos seguros ou limitações de hardware, isto terá um rótulo diferente para preservar as credenciais do AWS IoT. Se o seu dispositivo for compatível com o pré-provisionamento com uma chave RSA, forneça este rótulo. Quando PreProvisioned é definido como Sim em `device.json`, este rótulo,

`pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS`, ou ambos devem ser fornecidos.

`pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 da chave pública. Para dispositivos com elementos seguros ou limitações de hardware, isto terá um rótulo diferente para preservar as credenciais do AWS IoT. Se o seu dispositivo for compatível com o pré-provisionamento com uma chave RSA, forneça este rótulo. Quando PreProvisioned é definido como Sim em `device.json`, este rótulo, `pkcs11LabelPreProvisionedECDevicePublicKeyForTLS`, ou ambos devem ser fornecidos.

`pkcs11LabelPreProvisionedRSADeviceCertificateForTLS`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 do certificado do dispositivo. Para dispositivos com elementos seguros ou limitações de hardware, isto terá um rótulo diferente para preservar as credenciais do AWS IoT. Se o seu dispositivo for compatível com o pré-provisionamento com uma chave RSA, forneça este rótulo. Quando PreProvisioned é definido como Sim em `device.json`, este rótulo, `pkcs11LabelPreProvisionedECDeviceCertificateForTLS`, ou ambos devem ser fornecidos.

`pkcs11LabelCodeVerifyKey`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 da chave de verificação de código. Se o seu dispositivo tiver suporte de armazenamento PKCS n.º 11 para o certificado JITP, a chave de verificação de código e o certificado-raiz, forneça este rótulo. Quando `pkcs11JITPCodeVerifyRootCertSupport` em `device.json` estiver definido como Sim, este rótulo deverá ser fornecido.

`pkcs11LabelJITPCertificate`

(Opcional) Este rótulo é usado para o rótulo PKCS n.º 11 do certificado JITP. Se o seu dispositivo tiver suporte de armazenamento PKCS n.º 11 para o certificado JITP, a chave de verificação de código e o certificado-raiz, forneça este rótulo. Quando `pkcs11JITPCodeVerifyRootCertSupport` em `device.json` estiver definido como Sim, este rótulo deverá ser fornecido.

Variáveis do IDT para FreeRTOS

Os comandos para compilar o código e atualizar o dispositivo podem exigir conectividade ou outras informações sobre seus dispositivos para que possam ser executados. O AWS IoT Device Tester permite referenciar informações de dispositivo em comandos de atualização ou compilação usando JsonPath. Ao usar expressões JsonPath simples, você pode buscar as informações necessárias especificadas no arquivo `device.json`.

Variáveis de caminho

O IDT para FreeRTOS define as seguintes variáveis de caminho que podem ser usadas em linhas de comando e arquivos de configuração:

`{{testData.sourcePath}}`

Expande o caminho do código-fonte. Se você usar essa variável, ela deverá ser usada nos comandos de compilação e atualização.

`{{device.connectivity.serialPort}}`

Expande à porta serial.

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

Expande o número de série do dispositivo.

`{{config.idtRootPath}}`

Expande-se até o caminho-raiz do AWS IoT Device Tester.

Use a interface de usuário do IDT para FreeRTOS para executar o pacote de qualificação do FreeRTOS 2.0 (FRQ 2.0)

AWS IoT Device Tester para FreeRTOS (IDT for FreeRTOS) inclui uma interface de usuário (UI) baseada na web na qual você pode interagir com o aplicativo de linha de comando do IDT e os arquivos de configuração relacionados. Você usa a interface do IDT para FreeRTOS para criar uma nova configuração ou modificar uma existente para seu dispositivo. Você também pode usar a interface do usuário para chamar a aplicação IDT e executar os testes do FreeRTOS em seu dispositivo.

Para obter informações sobre o uso da linha de comando para executar os testes de qualificação, consulte Preparação para testar sua placa de microcontrolador pela primeira vez.

Esta seção descreve os pré-requisitos da interface do usuário do IDT para FreeRTOS e como executar testes de qualificação a partir da interface do usuário.

Tópicos

Pré-requisitos

Para executar testes por meio da interface do usuário AWS IoT Device Tester (IDT) for FreeRTOS, você deve preencher os pré-requisitos na página do IDT FreeRTOS Qualification (Pré-requisitosFRQ) 2.x.

Configurar credenciais do AWS

Você deve configurar suas credenciais de usuário do IAM para o AWS usuário que você criou. Criação e configuração de uma conta da AWS Você pode especificar suas credenciais de uma das seguintes formas:

- Em um arquivo de credenciais
- Como variáveis de ambiente

Configurar AWS credenciais com um arquivo de credenciais

O IDT usa o mesmo arquivo de credenciais que a AWS CLI. Para obter mais informações, consulte Arquivos de configuração e credenciais.

O local do arquivo de credenciais varia de acordo com o sistema operacional que você usa:

- macOS e Linux – `~/.aws/credentials`
- Windows: `C:\Users\UserName\.aws\credentials`

Adicione suas AWS credenciais ao `credentials` arquivo no seguinte formato:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

> **Note**
>
> Se você não usar o `default` AWS perfil, deverá especificar o nome do perfil na interface de usuário do IDT para FreeRTOS. Para obter mais informações sobre perfis, consulte Perfis nomeados.

Configurar AWS credenciais com variáveis de ambiente

As variáveis de ambiente são variáveis mantidas pelo sistema operacional e usadas pelos comandos do sistema. Elas não são salvas se você fechar a sessão SSH. A interface de usuário do IDT para FreeRTOS usa as variáveis de ambiente `AWS_SECRET_ACCESS_KEY` e `AWS_ACCESS_KEY_ID` para armazenar suas credenciais. AWS

Para definir essas variáveis no Linux, macOS ou Unix, use :export

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

Para definir essas variáveis no Windows, use :set

```
set AWS_ACCESS_KEY_ID=your_access_key_id
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

Abrir a interface de usuário do IDT para FreeRTOS

Como abrir a interface de usuário do IDT para FreeRTOS

1. Baixe um IDT compatível para a versão do FreeRTOS. Em seguida, extraia o arquivo baixado em um diretório para o qual você tenha permissões de leitura e gravação.
2. Navegue até o diretório da instalação do IDT para FreeRTOS:

   ```
   cd devicetester-extract-location/bin
   ```
3. Execute o comando a seguir para abrir a interface de usuário do IDT para FreeRTOS:

Linux

```
.devicetester_ui_linux_x86-64
```

Windows

```
./devicetester_ui_win_x64-64
```

macOS

```
./devicetester_ui_mac_x86-64
```

> **Note**
>
> No macOS, para permitir que seu sistema execute a interface do usuário, acesse Preferências do sistema -> Segurança e privacidade. Ao executar os testes, é aconselhável repeti-lo três vezes.

A interface de usuário do IDT para FreeRTOS é aberta em seu navegador padrão. As três versões principais mais recentes dos seguintes navegadores são compatíveis com a interface do usuário:

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

> **Note**
>
> Para obter uma melhor experiência, recomendamos o Google Chrome ou o Mozilla Firefox para acessar a interface de usuário do IDT para FreeRTOS. O Microsoft Internet Explorer não é compatível com a interface do usuário.

> **Important**
>
> Você deve configurar suas AWS credenciais antes de abrir a interface do usuário. Se você não configurou suas credenciais, feche a janela do navegador da interface de usuário do IDT para FreeRTOS, siga as etapas em Configurar credenciais do AWS e reabra a interface de usuário do IDT para FreeRTOS.

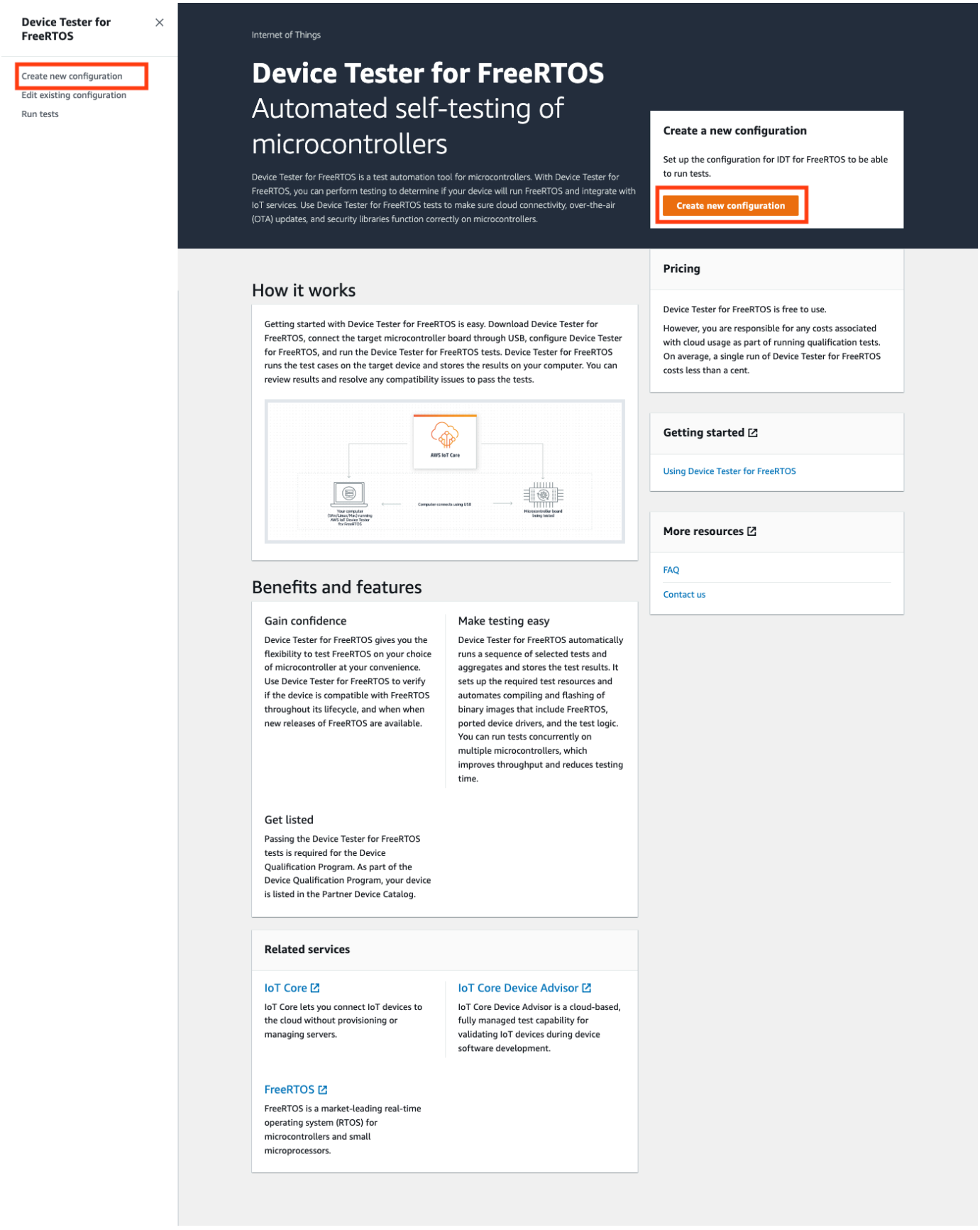
Criar uma configuração

Se você for um usuário iniciante, deverá criar uma nova configuração para definir os arquivos de configuração JSON que o IDT para FreeRTOS exige para executar testes. Em seguida, você pode executar testes ou modificar a configuração criada.

Para obter exemplos dos arquivos `config.json`, `device.json` e `userdata.json`, consulte Preparação para testar sua placa de microcontrolador pela primeira vez.

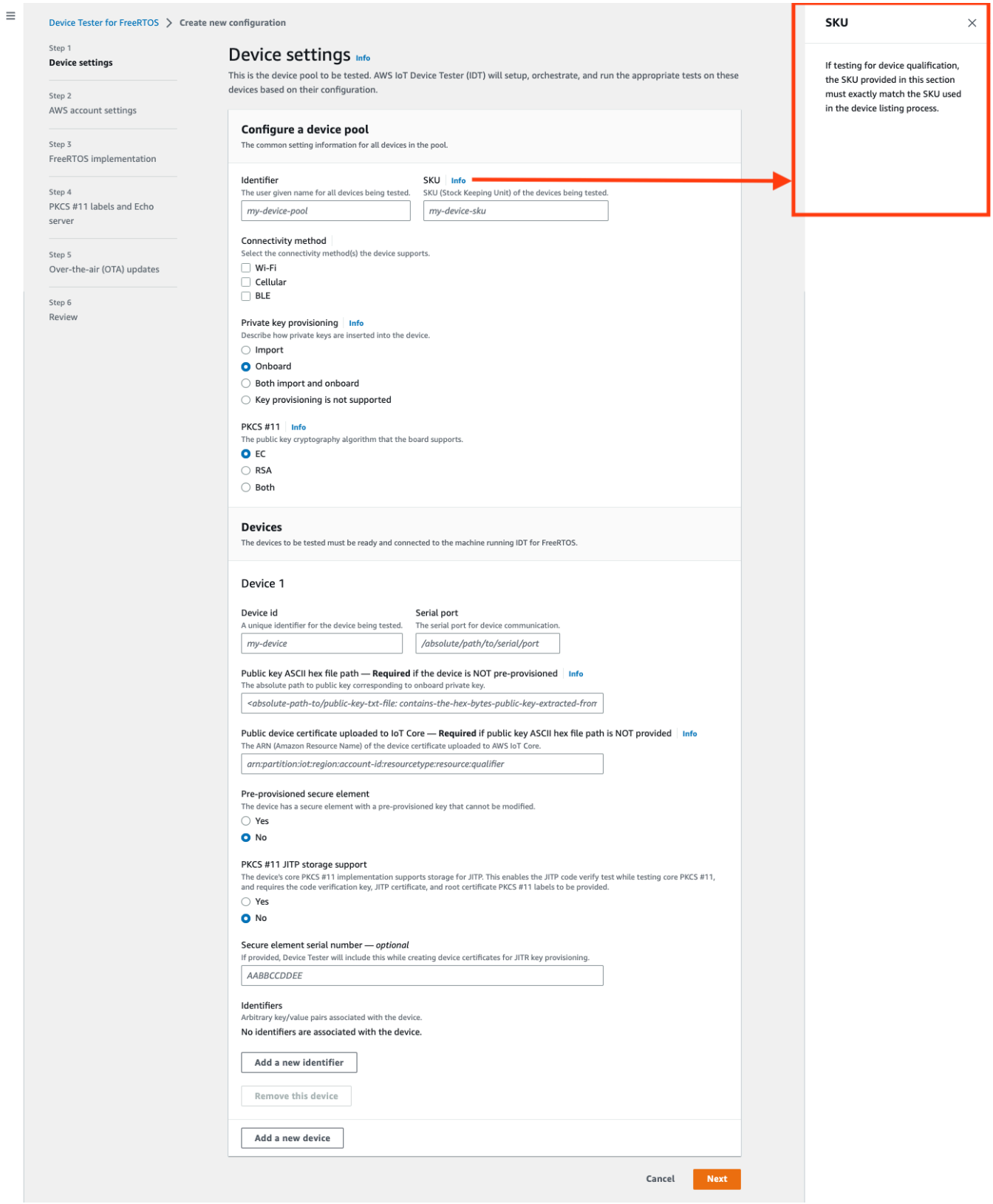
Como criar uma configuração

1. Na interface de usuário do IDT para FreeRTOS, abra o menu de navegação e escolha Criar nova configuração.

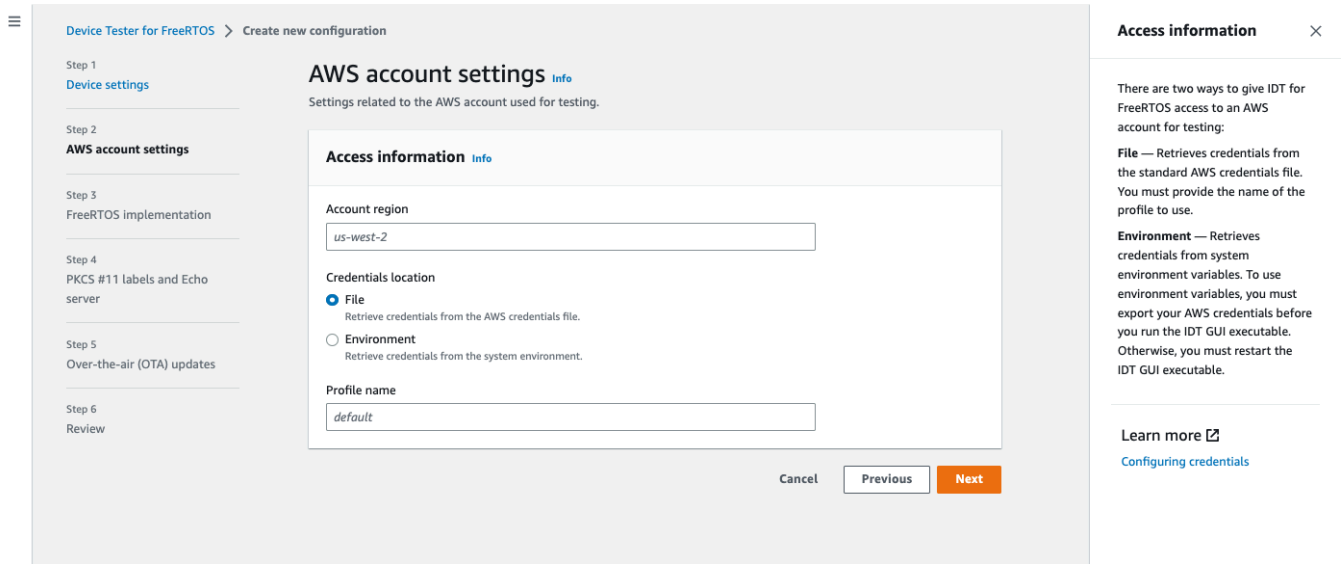

Device Tester for FreeRTOS
Create new configuration
Edit existing configuration
Run tests
Internet of Things
Device Tester for FreeRTOS
Automated self-testing of microcontrollers
Device Tester for FreeRTOS is a test automation tool for microcontrollers. With Device Tester for FreeRTOS, you can perform 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 o assistente de configuração para inserir as configurações do IDT usadas para executar os testes de qualificação. O assistente define as seguintes configurações nos arquivos de configuração JSON localizados no diretório *devicetester-extract-location*/config.

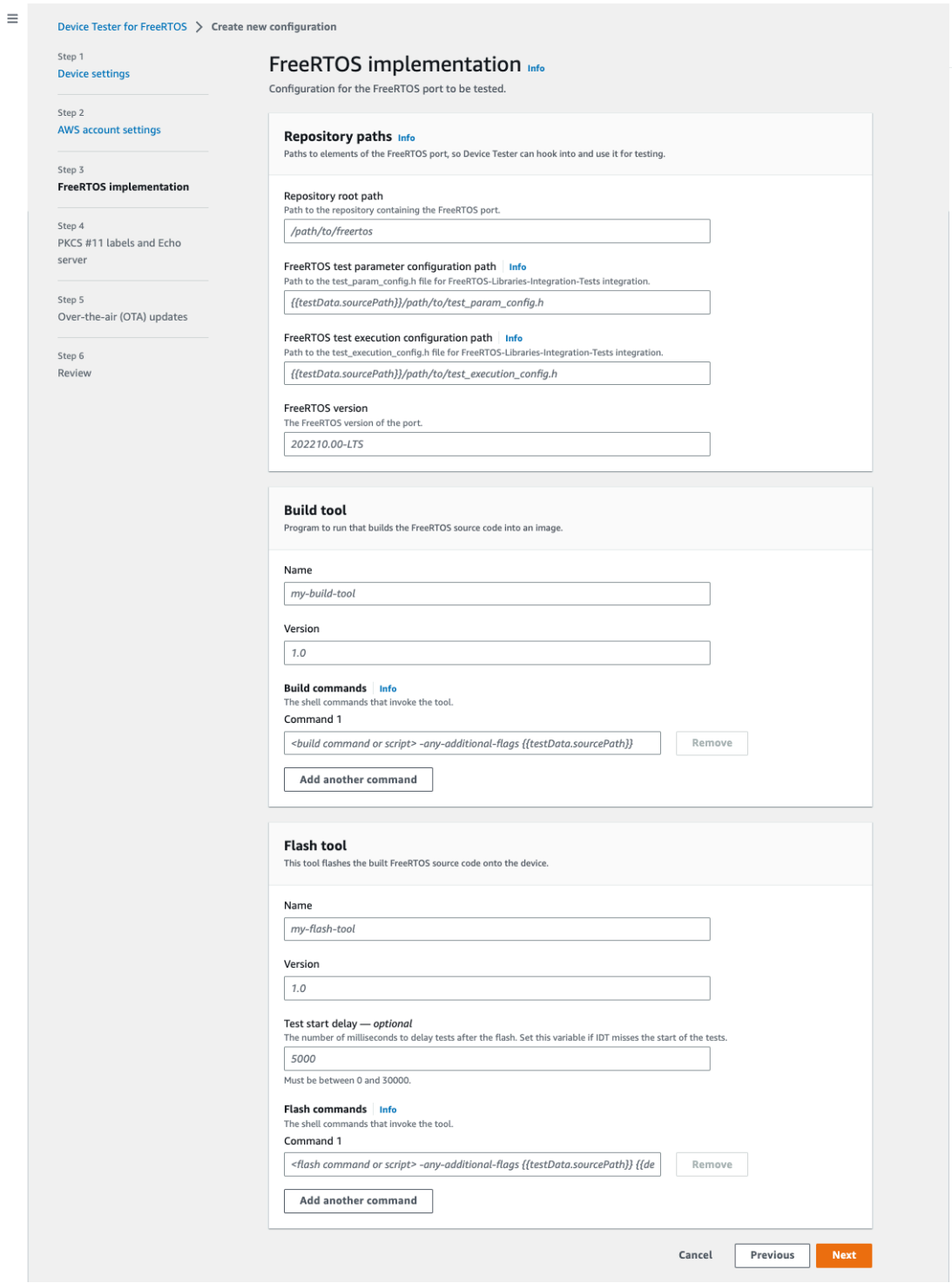
   - Configurações de dispositivos: as configurações do grupo de dispositivos para os dispositivos a serem testados. Essas configurações são definidas nos campos `id` e `sku` no bloco do grupo de dispositivos no arquivo `config.json`.


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.

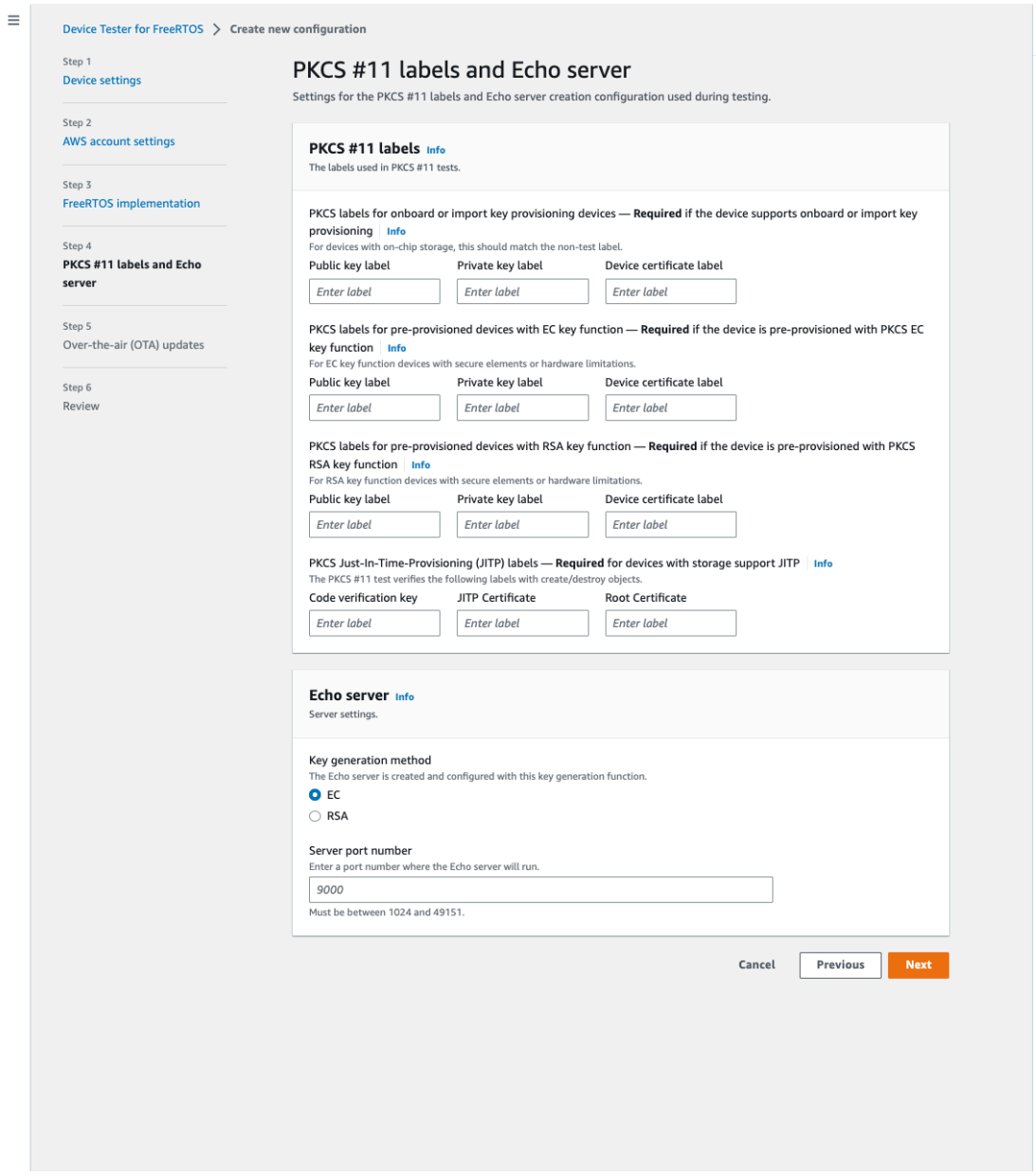
- AWS configurações da conta — As Conta da AWS informações que o IDT para FreeRTOS usa para AWS criar recursos durante as execuções de teste. Essas configurações são definidas no arquivo `config.json`.



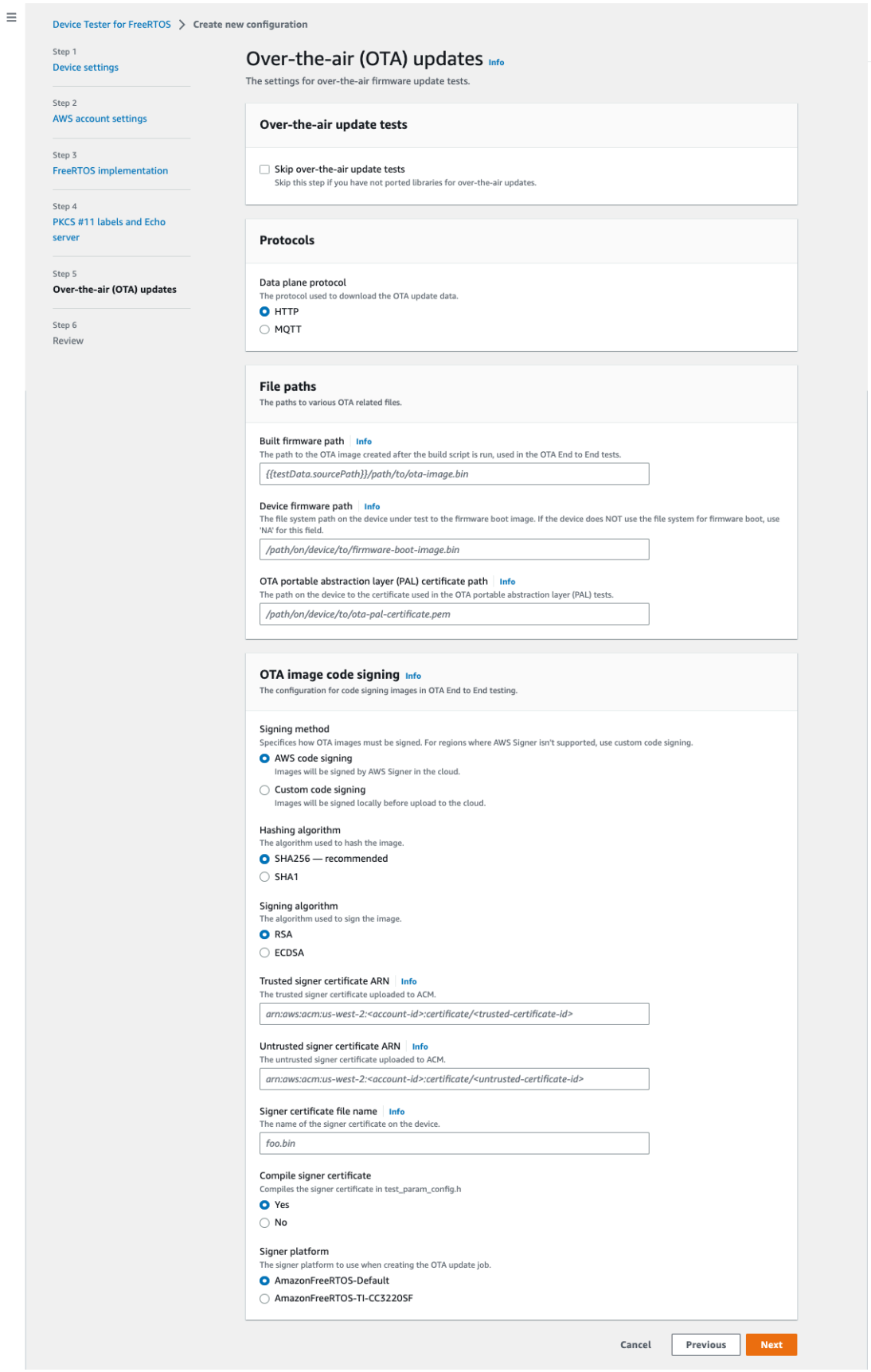
- Implementação do FreeRTOS: o caminho absoluto para o repositório e o código transferido do FreeRTOS e a versão do FreeRTOS na qual você deseja executar o FRQ do IDT. Os caminhos para os arquivos de cabeçalho de configuração de execução e parâmetros do `FreeRTOS-Libraries-Integration-Tests` GitHub repositório. Compilar e instalar: os comandos para seu hardware que permitem que o IDT compile e instale testes em sua placa automaticamente. Essas configurações são definidas no arquivo `userdata.json`.



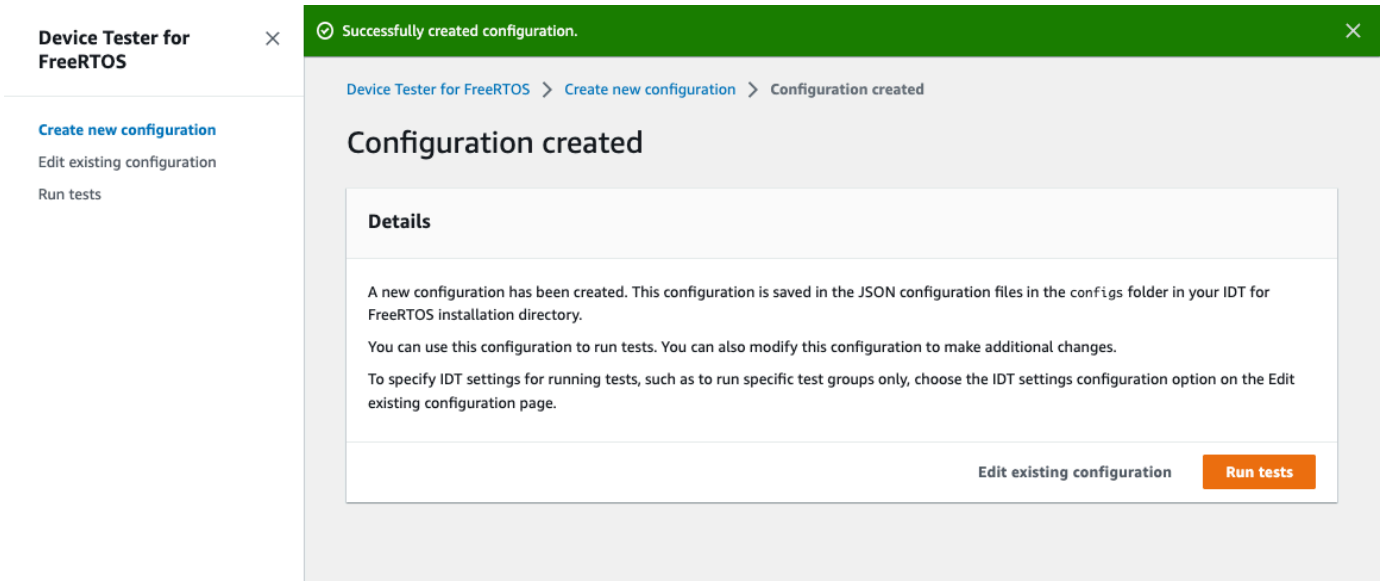
- Rótulos PKCS #11 e servidor echo: os rótulos PKCS #11 que correspondem às chaves provisionadas em seu hardware com base na funcionalidade e no método de provisionamento de chaves. As configurações do servidor echo para os testes da Interface de transporte. Essas configurações são definidas nos arquivos `userdata.json` e `device.json`.



- Atualizações O ver-the-air (OTA) — As configurações que controlam os testes de funcionalidade do OTA. Essas configurações são definidas no bloco `features` dos arquivos `device.json` e `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. Na página Revisar, verifique suas informações de configuração.



Depois de concluir a revisão da configuração, para executar os testes de qualificação, escolha Executar testes.

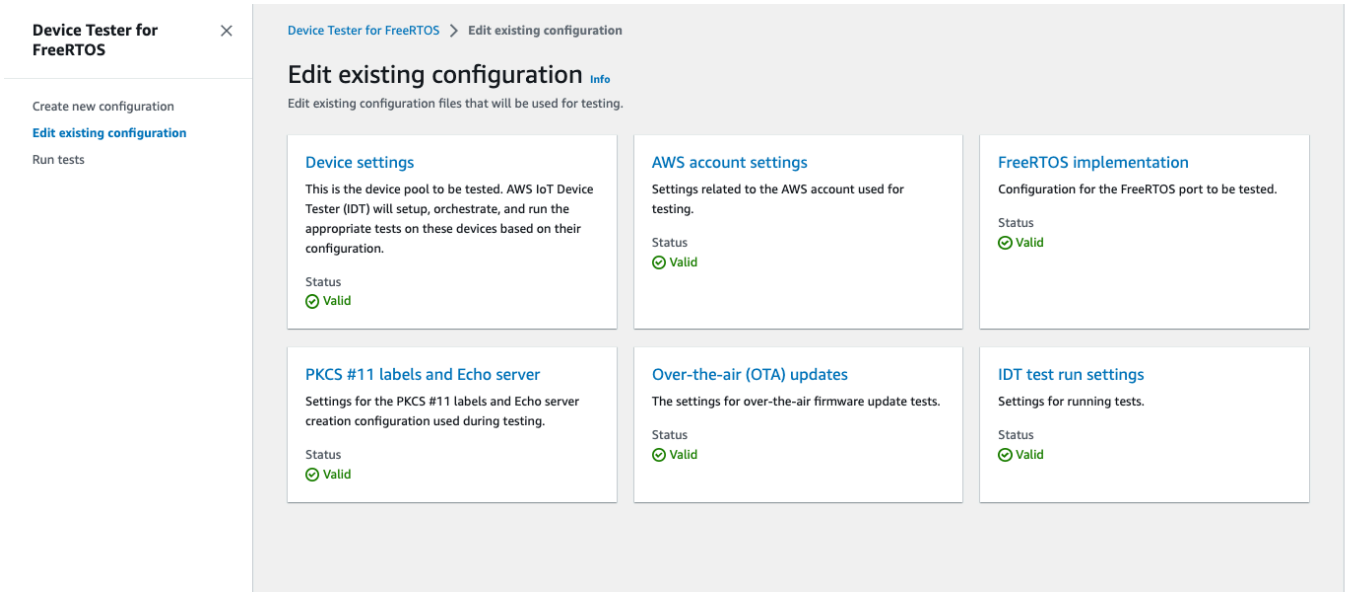
Modificar uma configuração existente

Se você já configurou arquivos de configuração do IDT para FreeRTOS, você pode usar a interface de usuário do IDT para FreeRTOS para modificar sua configuração existente. Os arquivos de configuração existentes devem estar localizados no diretório *devicetester-extract-location*`/config`.

Como criar uma configuração

1. Na interface de usuário do IDT para FreeRTOS, abra o menu de navegação e escolha Editar configuração existente.

   O painel de configuração exibe informações sobre suas configurações existentes. Se uma configuração estiver incorreta ou indisponível, o status dessa configuração será `Error validating configuration`.

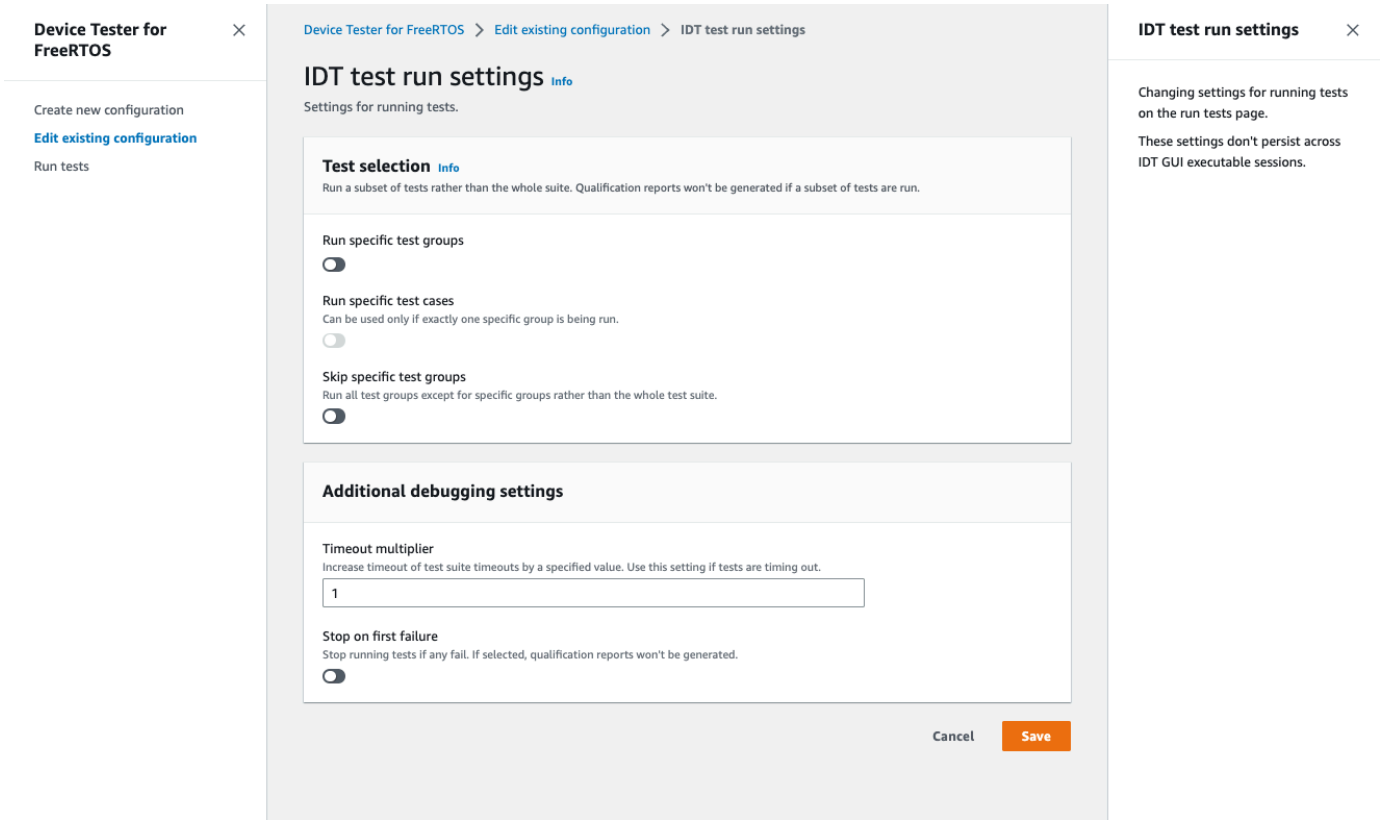


2. Para modificar uma configuração existente, conclua estas etapas:

   a. Escolha o nome de uma configuração para abrir sua página de ajustes.

   b. Modifique as configurações e escolha Salvar para regenerar o arquivo de configuração correspondente.

3. Para modificar as configurações de execução de teste do IDT para FreeRTOS, escolha as configurações de execução de teste do IDT na visualização de edição:



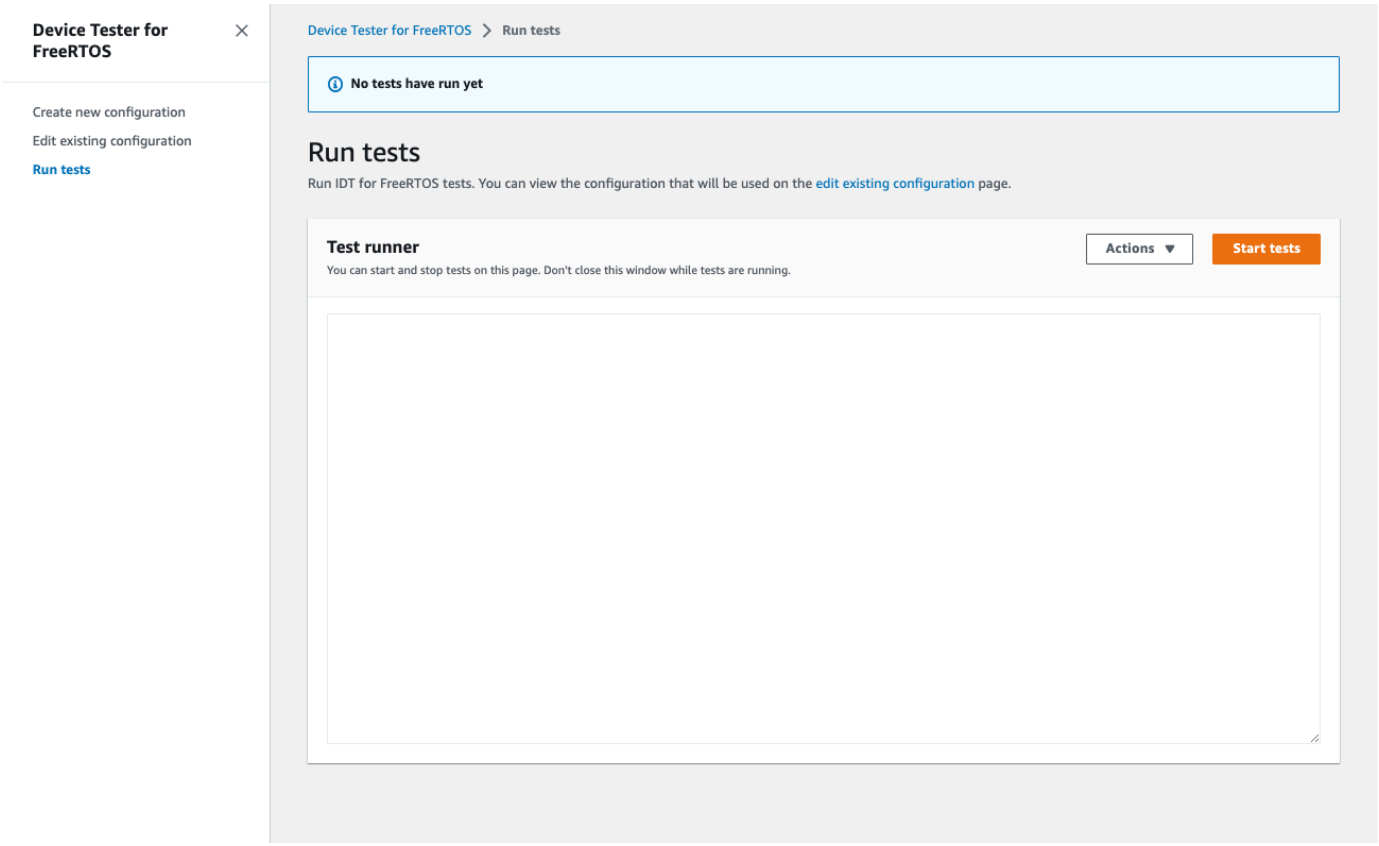
Depois de concluir a modificação da configuração, verifique se todas as configurações passaram pela validação. Se o status de cada configuração for `Valid`, será possível executar seus testes de qualificação com essa configuração.

Execute testes de qualificação

Após criar uma configuração para a interface de usuário do IDT para FreeRTOS, é possível executar os testes de qualificação.

Como executar testes de qualificação

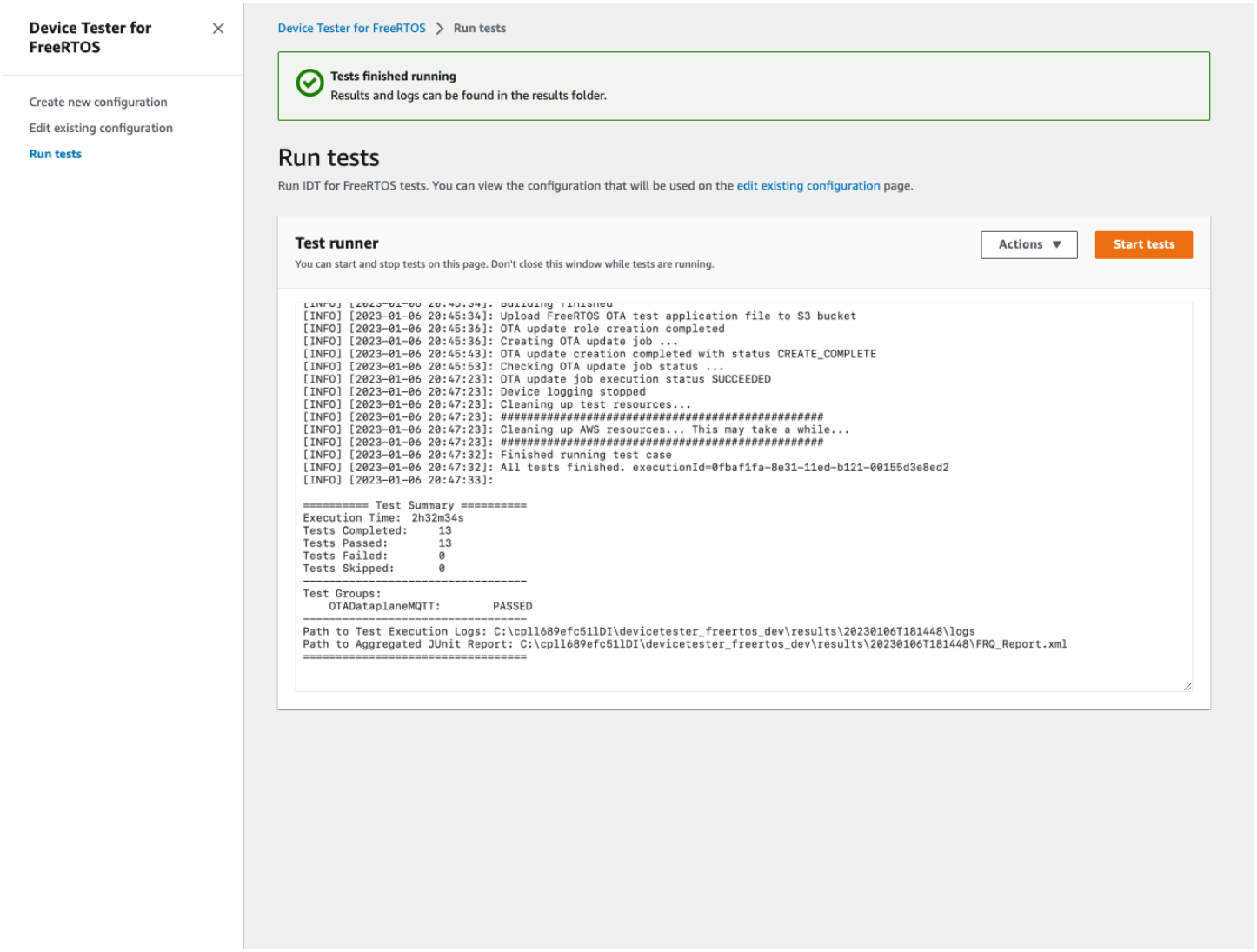
1. No menu de navegação, escolha Executar testes.
2. Escolha Iniciar testes para iniciar a execução do teste. Por padrão, todos os testes aplicáveis são executados para a configuração do seu dispositivo. O IDT para FreeRTOS gera um relatório de qualificação quando todos os testes terminam.



O IDT para FreeRTOS executa os testes de qualificação. Em seguida, ele exibe o resumo da execução do teste e todos os erros no console do Executor de teste. Depois que a execução do teste for concluída, você poderá visualizar os resultados e os logs do teste nos seguintes locais:

- Os resultados dos testes estão localizados no diretório *devicetester-extract-location*/`results/`*execution-id*.
- Os logs de teste estão localizados no diretório *devicetester-extract-location*/`results/`*execution-id*`/logs`.

Para obter mais informações sobre os resultados de teste e logs, consulte Noções básicas de resultados e logs.



Como executar o pacote de qualificação do FreeRTOS 2.0

Use o AWS IoT Device Tester para FreeRTOS executável para interagir com o IDT para FreeRTOS. Os exemplos de linha de comando a seguir mostram como executar os testes de qualificação para um grupo de dispositivos (um conjunto de dispositivos idênticos).

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
```

Executa um conjunto de testes em um grupo de dispositivos. O arquivo `userdata.json` deve estar localizado no diretório *`devicetester_extract_location`*`/devicetester_freertos_`*`[win|mac|linux]`*`/configs/`.

> **Note**
>
> Se estiver executando o IDT para FreeRTOS no Windows, use barras (/) para especificar o caminho até o arquivo `userdata.json`.

Use o seguinte comando para executar um grupo de testes específico:

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

Os parâmetros `suite-id` e `pool-id` são opcionais se você estiver executando um único conjunto de testes em um único grupo de dispositivos (ou seja, você tem apenas um grupo de dispositivos definido no arquivo `device.json`).

Use o seguinte comando para executar um caso de teste em um grupo de testes:

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

Você pode usar o comando `list-test-cases` para listar os casos de teste em um grupo de testes.

Opções de linhas de comando do IDT para FreeRTOS

group-id

(Opcional) Os grupos de testes a serem executados, como uma lista separada por vírgulas. Se não for especificado, o IDT executará todos os grupos de testes no conjunto de testes.

pool-id

(Opcional) O grupo de dispositivos a ser testado. Isso será necessário se você definir vários grupos de dispositivos no `device.json`. Se você tiver apenas um grupo de dispositivos, poderá omitir essa opção.

suite-id

(Opcional) A versão do conjunto de testes a ser executada. Se não for especificada, o IDT usará a versão mais recente no diretório de testes em seu sistema.

test-id

(Opcional) Os testes a serem executados, como uma lista separada por vírgulas. Se especificado, `group-id` deve especificar um único grupo.

Example

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

h

Use a opção de ajuda para saber mais sobre as opções de `run-suite`.

Example

Exemplo

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

Comandos do IDT para FreeRTOS

O comando do IDT para FreeRTOS é compatível com as seguintes operações:

IDT v4.5.2 and later

`help`

Relaciona as informações sobre o comando especificado.

`list-groups`

Lista os grupos em um determinado conjunto.

`list-suites`

Lista os conjuntos disponíveis.

`list-supported-products`

Lista os produtos compatíveis e as versões do conjunto de testes.

`list-supported-versions`

Lista as versões do FreeRTOS e do pacote de teste compatíveis com a versão atual do IDT.

`list-test-cases`

Lista os casos de teste em um grupo especificado.

`run-suite`

Executa um conjunto de testes em um grupo de dispositivos.

Use a opção `--suite-id` para especificar uma versão do conjunto de testes, ou omiti-la para usar a versão mais recente em seu sistema.

Use o `--test-id` para executar um caso de teste individual.

Example

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

> **Note**
>
> Começando com o IDT v3.0.0, o IDT verifica se há conjuntos de testes mais recentes online. Para obter mais informações, consulte Versões do conjunto de testes.

Noções básicas de resultados e logs

Esta seção descreve como visualizar e interpretar os resultados de relatórios e logs do IDT.

Visualização de resultados

Enquanto em execução, o IDT grava erros no console, arquivos de log e relatórios de teste. Depois de concluir o pacote de teste de qualificação, O IDT gravará um resumo da execução de

teste no console e gerará dois relatórios de teste. Esses relatórios podem ser encontrados em *devicetester-extract-location*/results/*execution-id*/. Ambos os relatórios capturam os resultados da execução do pacote de teste de qualificação.

O awsiotdevicetester_report.xml é o relatório de teste de qualificação que você envia para a AWS para listar o dispositivo na solução AWS Partner Device Catalog. O relatório contém os seguintes elementos:

- A versão do IDT para FreeRTOS.
- A versão do FreeRTOS que foi testada.
- Os atributos do FreeRTOS que são compatíveis com o dispositivo com base nos testes aprovados.
- A SKU e o nome do dispositivo especificados no arquivo device.json.
- Os recursos do dispositivo especificados no arquivo device.json.
- O resumo agregado dos resultados de casos de teste.
- Um detalhamento dos resultados de casos de teste por bibliotecas que foram testadas com base nos recursos do dispositivo.

O FRQ_Report.xml é um relatório no formato padrão JUnit XML. É possível integrá-lo a plataformas de CI/CD, como Jenkins, Bamboo e assim por diante. O relatório contém os seguintes elementos:

- Um resumo agregado dos resultados de casos de teste.
- Um detalhamento dos resultados de casos de teste por bibliotecas que foram testadas com base nos recursos do dispositivo.

Como interpretar os resultados do IDT para FreeRTOS

A seção de relatório em awsiotdevicetester_report.xml ou em FRQ_Report.xml lista os resultados dos testes que são executados.

A primeira tag XML <testsuites> contém o resumo geral da execução do teste. Por exemplo:

```
<testsuites name="FRQ results" time="5633" tests="184" failures="0"
errors="0" disabled="0">
```

Atributos usados na tag **<testsuites>**

`name`

O nome do conjunto de testes.

`time`

O tempo, em segundos, necessário para executar o conjunto de qualificação.

`tests`

O número de casos de teste executados.

`failures`

O número de casos de teste que foram executados, mas não foram aprovados.

`errors`

O número de casos de teste que o IDT para FreeRTOS não pôde executar.

`disabled`

Esse atributo não é usado e pode ser ignorado.

Se não houver falhas ou erros de casos de teste, seu dispositivo atende aos requisitos técnicos para executar o FreeRTOS e pode interoperar com serviços do AWS IoT. Se optar por listar o dispositivo na solução AWS Partner Device Catalog, poderá usar este relatório como evidência de qualificação.

Se houver falhas de caso de teste ou erros, você poderá identificar o caso de teste com falha analisando as tags XML `<testsuites>`. As tags XML `<testsuite>` dentro da tag `<testsuites>` mostram o resumo do resultado do caso de teste para um grupo de testes.

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

O formato é semelhante à tag `<testsuites>`, mas com um atributo chamado `skipped` que não é usado e pode ser ignorado. Dentro de cada tag XML `<testsuite>`, há tags `<testcase>` para cada um dos casos de teste que foram executados para um grupo de testes. Por exemplo:

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

Atributos usados na tag **`<awsproduct>`**

`name`

O nome do produto testado.

`version`

A versão do produto testado.

`features`

Os recursos validados. Recursos marcados como `required` são necessários para enviar sua placa para qualificação. O snippet a seguir mostra como essas informações aparecem no arquivo `awsiotdevicetester_report.xml`.

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

Os recursos marcados como `optional` não são necessários para qualificação. Os seguintes trechos mostram recursos opcionais.

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

Se não há falhas de teste ou erros nos atributos exigidos, seu dispositivo atende aos requisitos técnicos para executar o FreeRTOS e pode interoperar com serviços do AWS IoT. Se quiser listar o dispositivo no AWSPartner Device Catalog, poderá usar este relatório como evidência de qualificação.

Se houver falhas de teste ou erros, você poderá identificar o teste com falha analisando as tags XML `<testsuites>`. As tags XML `<testsuite>` dentro da tag `<testsuites>` mostram o resumo do resultado do teste para um grupo de testes. Por exemplo:

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

O formato é semelhante à tag `<testsuites>`, mas tem um atributo `skipped` que não é usado e pode ser ignorado. Dentro de cada tag XML `<testsuite>`, há tags `<testcase>` para cada teste executado para um grupo de testes. Por exemplo:

```
<testcase classname="FreeRTOSVersion" name="FreeRTOSVersion"></testcase>
```

Atributos usados na tag **<testcase>**

name

O nome do caso de teste.

attempts

O número de vezes que o IDT para FreeRTOS executou o caso de teste.

Quando um teste falha ou ocorre um erro, as tags <failure> ou <error> são adicionadas à tag <testcase> com informações para a solução de problemas. Por exemplo:

```
<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 obter mais informações, consulte Solução de problemas.

Visualizar logs do

É possível encontrar os logs que o IDT para FreeRTOS gera ao executar testes em *devicetester-extract-location*/results/*execution-id*/logs. Dois conjuntos de logs são gerados:

- test_manager.log

  Contém logs gerados de IDT para FreeRTOS (por exemplo, logs relacionados à configuração e geração de relatórios).

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

  O arquivo de log para um caso de teste, incluindo a saída do dispositivo em teste. O arquivo de log é nomeado de acordo com o grupo de teste e com o caso de teste que foi executado.

Use o IDT com o pacote de qualificação do FreeRTOS 1.0 (FRQ 1.0)

> **Important**
>
> Em outubro de 2022, AWS IoT Device Tester o AWS IoT FreeRTOS Qualification (FRQ) 1.0 não gera relatórios de qualificação assinados. Você não pode qualificar novos dispositivos AWS IoT FreeRTOS para serem listados no Catálogo de Dispositivos Parceiros por meio AWS do Programa de Qualificação de Dispositivos usando AWS as versões IDT FRQ 1.0. Embora você não possa qualificar dispositivos FreeRTOS usando o IDT FRQ 1.0, você pode continuar testando seus dispositivos FreeRTOS com o FRQ 1.0. Recomendamos que você use o IDT FRQ 2.0 para qualificar e listar dispositivos FreeRTOS no AWS Partner Device Catalog.

Você pode usar o IDT para qualificação do FreeRTOS para verificar se o sistema operacional do FreeRTOS funciona localmente em seu dispositivo e pode se comunicar com ele. AWS IoT Especificamente, ele verifica se as interfaces da camada de portabilidade para as bibliotecas do FreeRTOS estão implementadas corretamente. Ele também realiza end-to-end testes com AWS IoT Core. Por exemplo, ele verifica se a placa pode enviar e receber mensagens MQTT e processá-las corretamente. Os testes executados pelo IDT para FreeRTOS são definidos no repositório do FreeRTOS. GitHub

Os testes são executados como aplicações incorporadas que são atualizadas na placa. As imagens binárias da aplicação incluem o FreeRTOS, as interfaces do FreeRTOS obtidas por portabilidade do fornecedor de semicondutores e os drivers de dispositivos da placa. A finalidade dos testes é verificar se as interfaces do FreeRTOS obtidas por portabilidade funcionam corretamente em cima dos drivers de dispositivo.

O IDT for FreeRTOS gera relatórios de teste que você pode enviar AWS IoT para adicionar seu hardware ao AWS Partner Device Catalog. Para obter mais informações, consulte Programa de Qualificação de Dispositivos da AWS.

O IDT para FreeRTOS é executado em um computador host (Windows, macOS ou Linux) que está conectado à placa a ser testada. O IDT executa casos de teste e agrega os resultados. Ele também fornece uma interface de linha de comando para gerenciar a execução do teste.

Além de testar dispositivos, o IDT for FreeRTOS cria recursos (por exemplo, AWS IoT coisas, grupos do FreeRTOS, funções do Lambda etc.) para facilitar o processo de qualificação. Para criar esses recursos, o IDT para FreeRTOS usa AWS as credenciais configuradas no para fazer chamadas de API em `config.json` seu nome. Esses recursos são provisionados em vários momentos durante o teste.

Ao executar o IDT para FreeRTOS em seu computador host, ele executa as seguintes etapas:

1. Carrega e valida o dispositivo e configuração de credenciais.
2. Executa testes selecionados com os recursos locais e de nuvem necessários.
3. Remove recursos locais e de nuvem.
4. Gera relatórios de testes que indicam se a placa passou nos testes necessários para a qualificação.

Tópicos

Pré-requisitos

Esta seção descreve os pré-requisitos para testar microcontroladores com. AWS IoT Device Tester

Fazer download do FreeRTOS

Você pode baixar uma versão do FreeRTOS com o seguinte GitHubcomando:

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

em que <FREERTOS_RELEASE_VERSION> é uma versão do FreeRTOS (por exemplo, 202007.00) correspondente a uma versão do IDT listada em Versões compatíveis do AWS IoT Device Tester para FreeRTOS. Isso garante que você tenha o código-fonte completo, incluindo submódulos, e esteja usando a versão correta do IDT para sua versão de FreeRTOS e vice-versa.

O Windows tem uma limitação de comprimento de caminho de 260 caracteres. A estrutura de caminho do FreeRTOS tem muitos níveis; portanto, se você estiver usando o Windows, mantenha os caminhos de arquivo dentro do limite de 260 caracteres. Por exemplo, clone o FreeRTOS para `C:\FreeRTOS` em vez de `C:\Users\username\programs\projects\myproj\FreeRTOS\`.

Considerações para a qualificação LTS (qualificação para FreeRTOS que usa bibliotecas LTS)

- Para que seu microcontrolador seja designado como compatível com versões de FreeRTOS baseadas em suporte de longo prazo (LTS) no AWS Partner Device Catalog, você deve fornecer um arquivo manifesto. Para obter mais informações, consulte a Lista de verificação de qualificação do FreeRTOS no Guia de qualificação do FreeRTOS.
- Para validar se seu microcontrolador suporta versões baseadas em LTS do FreeRTOS e qualificá-lo para envio ao AWS Partner Device Catalog, você deve usar (AWS IoT Device Tester IDT) com a suíte de testes FreeRTOS Qualification (FRQ) versão v1.4.x.
- O suporte para versões baseadas em LTS do FreeRTOS está limitado à versão 202012.xx do FreeRTOS.

Faça download do IDT para FreeRTOS

Cada versão do FreeRTOS tem uma versão correspondente do IDT para o FreeRTOS para realizar testes de qualificação. Baixe a versão adequada do IDT para FreeRTOS em Versões compatíveis do AWS IoT Device Tester para FreeRTOS.

Extraia o IDT para FreeRTOS em um local no sistema de arquivos onde existam permissões de leitura e gravação. Como o Microsoft Windows tem um limite de caracteres para o comprimento do caminho, extraia o IDT para FreeRTOS em um diretório raiz, como `C:\` ou `D:\`.

> **Note**
>
> Não é recomendável que vários usuários executem o IDT em um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Isso pode resultar em falhas ou corrupção de dados. Recomendamos extrair o pacote IDT para uma unidade local.

Crie e configure uma AWS conta

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

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

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .

2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

AWS IoT Device Tester política gerenciada

A política `AWSIoTDeviceTesterForFreeRTOSFullAccess` gerenciada contém as seguintes AWS IoT Device Tester permissões para verificação de versão, recursos de atualização automática e coleção de métricas.

- `iot-device-tester:SupportedVersion`

Concede AWS IoT Device Tester permissão para buscar a lista de produtos compatíveis, suítes de teste e versões do IDT.

- `iot-device-tester:LatestIdt`

  Concede AWS IoT Device Tester permissão para obter a versão mais recente do IDT disponível para download.

- `iot-device-tester:CheckVersion`

  Concede AWS IoT Device Tester permissão para verificar a compatibilidade de versões do IDT, suítes de teste e produtos.

- `iot-device-tester:DownloadTestSuite`

  Concede AWS IoT Device Tester permissão para baixar atualizações da suíte de testes.

- `iot-device-tester:SendMetrics`

  Concede AWS permissão para coletar métricas sobre o uso AWS IoT Device Tester interno.

(Opcional) Instale o AWS Command Line Interface

Talvez você prefira usar o AWS CLI para realizar algumas operações. Se não tiver a AWS CLI instalada, siga as instruções em Instalar a AWS CLI.

Configure o AWS CLI para a AWS região que você deseja usar executando a aws configure partir de uma linha de comando. Para obter informações sobre as AWS regiões que oferecem suporte ao IDT para FreeRTOS,AWS consulte Regiões e endpoints. Para obter mais informações sobre aws configure, consulte Configuração rápida com o aws configure.

Preparação para testar sua placa de microcontrolador pela primeira vez

Você pode usar o IDT para FreeRTOS para testar conforme transfere as interfaces do FreeRTOS. Depois de portar as interfaces FreeRTOS para os drivers de dispositivo da sua placa, você pode executar os testes AWS IoT Device Tester de qualificação na placa microcontroladora.

Adicionar camadas de transferência da biblioteca

Para fazer a portabilidade do FreeRTOS para seu dispositivo, siga as instruções no Guia de portabilidade do FreeRTOS.

Configure suas AWS credenciais

Você precisa configurar suas AWS credenciais para se AWS IoT Device Tester comunicar com a AWS nuvem. Para obter mais informações, consulte Configurar as credenciais e a região da AWS para desenvolvimento. AWS As credenciais válidas devem ser especificadas no arquivo *devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/config.json de configuração.

Crie um grupo de dispositivos no IDT para FreeRTOS

Os dispositivos a serem testados são organizados em grupos de dispositivos. Cada grupo de dispositivos consiste em um ou mais dispositivos idênticos. Você pode configurar o IDT para FreeRTOS para testar um único dispositivo em um grupo ou vários dispositivos em um grupo. Para acelerar o processo de qualificação, o IDT para FreeRTOS pode testar dispositivos com as mesmas especificações em paralelo. Ele usa o método round-robin para executar um grupo de testes diferente em cada dispositivo de um grupo de dispositivos.

Você pode adicionar um ou mais dispositivos a um grupo de dispositivos editando a seção `devices` do modelo `device.json` na pasta `configs`.

> **Note**
>
> Todos os dispositivos no mesmo grupo devem ser da mesma especificação técnica e SKU.

Para habilitar as compilações paralelas do código-fonte para diferentes grupos de teste, o IDT para FreeRTOS copia o código-fonte para uma pasta de resultados dentro da pasta onde o IDT para FreeRTOS foi extraído. O caminho do código-fonte no comando de compilação ou atualização deve ser referenciado por meio da variável `testdata.sourcePath` ou `sdkPath`. O IDT para FreeRTOS substitui esta variável por um caminho temporário do código-fonte copiado. Para obter mais informações, consulte, Variáveis do IDT para FreeRTOS.

Veja a seguir um exemplo do arquivo `device.json` usado para criar um grupo de dispositivos com vários 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"
            }
          ]
        }
      ]
  }
]
```

Os seguintes recursos são usados no arquivo `device.json`:

`id`

Um ID alfanumérico definido pelo usuário que identifica exclusivamente um grupo de dispositivos. Os dispositivos que pertencem a um grupo devem ser do mesmo tipo. Quando um conjunto de testes está em execução, os dispositivos do grupo são usados para paralelizar a carga de trabalho.

`sku`

Um valor alfanumérico que identifica exclusivamente a placa sendo testada. A SKU é usada para rastrear as placas qualificadas.

> **Note**
>
> Se você quiser listar sua placa no AWS Partner Device Catalog, o SKU especificado aqui deve corresponder ao SKU que você usa no processo de listagem.

`features`

Uma matriz que contém os recursos suportados pelo dispositivo. AWS IoT Device Tester usa essas informações para selecionar os testes de qualificação a serem executados.

Os valores compatíveis são:

`TCP/IP`

Indica se a placa oferece suporte a uma pilha TCP/IP e se ela é compatível no chip (MCU) ou descarregada em outro módulo. O TCP/IP é necessário para qualificação.

`WIFI`

Indica se a placa tem recursos de Wi-Fi. Deve ser definido como `No`, se `Cellular` estiver definido como `Yes`.

`Cellular`

Indica se a placa tem recursos de celular. Deve ser definido como `No`, se `WIFI` estiver definido como `Yes`. Quando esse recurso estiver configurado como`Yes`, o FullSecureSockets teste será executado usando instâncias EC2 AWS t2.micro e isso poderá gerar custos adicionais para sua conta. Para obter mais informações, consulte Definição de preço do Amazon EC2.

`TLS`

Indica se sua placa oferece suporte a TLS. O TLS é necessário para qualificação.

`PKCS11`

Indica o algoritmo de criptografia de chave pública compatível com a placa. O PKCS11 é necessário para qualificação. Os valores compatíveis são `ECC`, `RSA`, `Both` e `No`. `Both` indica que a placa é compatível com os algoritmos `ECC` e `RSA`.

`KeyProvisioning`

Indica o método de gravação de um certificado de cliente X.509 confiável em sua placa. Os valores válidos são `Import`, `Onboard` e `No`. O provisionamento de chaves é necessário para a qualificação.

- Use `Import` se a sua placa permitir a importação de chaves privadas. O IDT criará uma chave privada e criará isso para o código-fonte do FreeRTOS.
- Use `Onboard` se a placa for compatível com a geração de chaves privadas integradas (por exemplo, se o dispositivo tiver um elemento seguro ou se você preferir gerar o seu próprio par de chaves de dispositivo e certificado). Adicione um elemento `secureElementConfig`

em cada uma das seções do dispositivo e coloque o caminho absoluto para o arquivo de chave pública no campo `publicKeyAsciiHexFilePath`.

- Se a placa não for compatível com o provisionamento de chaves, use `No`.

`OTA`

Indica se sua placa suporta a funcionalidade de atualização over-the-air (OTA). O recurso `OtaDataPlaneProtocol` indica a qual protocolo de plano de dados OTA o dispositivo oferece suporte. O atributo será ignorado se o dispositivo não oferecer suporte ao recurso OTA. Quando `"Both"` é selecionado, o tempo de execução do teste OTA é prolongado devido à execução de testes MQTT, HTTP e mistos.

> **Note**
>
> A partir do IDT v4.1.0, `OtaDataPlaneProtocol` aceita somente `HTTP` e `MQTT` como valores compatíveis.

`BLE`

Indica se a placa é compatível com Bluetooth Low Energy (BLE).

`devices.id`

Um identificador exclusivo, definido pelo usuário, para o dispositivo que está sendo testado.

`devices.connectivity.protocol`

O protocolo de comunicação usado para se comunicar com esse dispositivo. Valor compatível: `uart`.

`devices.connectivity.serialPort`

A porta serial do computador host usada para se conectar aos dispositivos que estão sendo testados.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

O caminho absoluto para o arquivo que contém a chave pública de bytes hexadecimal extraída da chave privada integrada.

Formato de exemplo:

```
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
```

Se sua chave pública estiver no formato .der, você pode codificar a chave pública em hexadecimal diretamente para gerar o arquivo hexadecimal.

Exemplo de comando para a chave pública .der para gerar o arquivo hexadecimal:

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

Se sua chave pública estiver no formato .pem, é possível extrair a parte codificada em base64, decodificá-la em formato binário e, em seguida, codificá-la em hexadecimal para gerar o arquivo hexadecimal.

Por exemplo, use esses comandos para gerar um arquivo hexadecimal para a chave pública .pem:

1. Retire a parte codificada em base64 da chave (retire o cabeçalho e o rodapé) e armazene-a em um arquivo, por exemplo, nomeie-a como `base64key`, execute este comando para convertê-la para o formato .der:

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

2. Execute o comando `xxd` para convertê-la para o formato hexadecimal.

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

`devices.secureElementConfig.SecureElementSerialNumber`

(Opcional) O número de série do elemento seguro. Forneça esse campo quando o número de série estiver impresso com a chave pública do dispositivo ao executar o projeto de demonstração/teste do FreeRTOS.

`devices.secureElementConfig.preProvisioned`

(Opcional) Defina como "Sim" se o dispositivo tiver um elemento seguro pré-provisionado com credenciais bloqueadas, que não possa importar, criar ou destruir objetos. Essa configuração tem

efeito apenas quando o `features` tem a `KeyProvisioning` definida como "Onboard", junto com `PKCS11` definida como "ECC".

`identifiers`

(Opcional) Uma matriz de pares de nome-valor arbitrários. Você pode usar esses valores nos comandos de compilação e atualização descritos na próxima seção.

Configuração de parâmetros de compilação, atualização e teste

Para que o IDT para FreeRTOS crie e atualize testes em sua placa automaticamente, você deve configurar o IDT para executar os comandos de compilação e atualização para seu hardware. As configurações de comando de compilação e atualização são definidas no modelo de arquivo `userdata.json` localizado na pasta `config`.

Configuração de parâmetros para testar dispositivos

As configurações de compilação, atualização e teste são feitas no arquivo `configs/userdata.json`. Oferecemos suporte à configuração do servidor echo carregando os certificados e chaves do cliente e do servidor no `customPath`. Para obter mais informações, consulte Configurando um servidor Echo no Guia de portabilidade do FreeRTOS. O seguinte exemplo de JSON mostra como você pode configurar o IDT para FreeRTOS para testar vários 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"
            }
        ]
    }
}
```

Veja a seguir uma lista dos recursos usados no `userdata.json`:

`sourcePath`

O caminho para a raiz do código-fonte do FreeRTOS transferido. Para testes paralelos com um SDK, o `sourcePath` pode ser definido usando o espaço reservado `{{userData.sdkConfiguration.path}}`. Por exemplo: .

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

vendorPath

O caminho para o código do FreeRTOS específico do fornecedor. Para testes em série, o `vendorPath` pode ser definido como um caminho absoluto. Por exemplo: .

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

Para testes paralelos, o `vendorPath` pode ser definido usando o espaço reservado `{{testData.sourcePath}}`. Por exemplo: .

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

A variável `vendorPath` só é necessária quando executada sem um SDK, caso contrário, ela pode ser removida.

> **Note**
>
> Ao executar testes em paralelo sem um SDK, o espaço reservado `{{testData.sourcePath}}` deve ser usado nos campos `vendorPath`, `buildTool`, `flashTool`. o executar o teste com um único dispositivo, caminhos absolutos devem ser usados nos campos `vendorPath`, `buildTool` e `flashTool`. Ao executar com um SDK, o espaço reservado `{{sdkPath}}` deve ser usado nos comandos `sourcePath`, `buildTool` e `flashTool`.

sdkConfiguration

Se você estiver qualificando o FreeRTOS com modificações na estrutura de arquivos e pastas além do necessário para a portabilidade, será necessário configurar as informações do SDK neste bloco. Se você não estiver qualificando com um Freertos transferido de dentro de um SDK, será necessário omitir esse bloco totalmente.

sdkConfiguration.name

O nome do SDK que você está usando com o FreeRTOS. Se você não estiver usando um SDK, todo o bloco `sdkConfiguration` deverá ser omitido.

`sdkConfiguration.version`

A versão do SDK que você está usando com o FreeRTOS. Se você não estiver usando um SDK, todo o bloco `sdkConfiguration` deverá ser omitido.

`sdkConfiguration.path`

O caminho absoluto para o diretório do SDK que contém o código do FreeRTOS. Se você não estiver usando um SDK, todo o bloco `sdkConfiguration` deverá ser omitido.

`buildTool`

O caminho completo para o script de compilação (.bat ou .sh) que contém os comandos para compilar seu código-fonte. Todas as referências ao caminho do código-fonte no comando build devem ser substituídas pela AWS IoT Device Tester variável `{{testdata.sourcePath}}` e as referências ao caminho do SDK devem ser substituídas por`{{sdkPath}}`. Use o `{{config.idtRootPath}}` espaço reservado para referenciar o caminho IDT absoluto ou relativo.

`testStartDelayms`

Especifica quantos milissegundos o executor de testes do FreeRTOS aguardará antes de começar a executar os testes. Isso pode ser útil se o dispositivo em teste começar a gerar informações de teste importantes antes que o IDT tenha a chance de se conectar e iniciar o registro em log devido à rede ou outra latência. O valor máximo permitido é 30.000 ms (30 segundos). Este valor é aplicável somente aos grupos de teste do FreeRTOS, e não aplicável a outros grupos de teste que não utilizam o executor de testes do FreeRTOS, como os testes OTA.

`flashTool`

O caminho completo para o script de atualização (.sh ou .bat) que contém os comandos de atualização para o dispositivo. Todas as referências ao caminho do código-fonte no comando de atualização devem ser substituídas pela variável `{{testdata.sourcePath}}` do IDT para FreeRTOS e todas as referências ao seu caminho do SDK devem ser substituídas pela variável`{{sdkPath}}` do IDT para FreeRTOS. Use o espaço reservado `{{config.idtRootPath}}` para fazer referência ao caminho absoluto ou relativo do IDT.

`buildImageInfo`

`testsImageName`

O nome do arquivo produzido pelo comando de compilação ao compilar testes da pasta *`freertos-source`*`/tests`.

`demosImageName`

O nome do arquivo produzido pelo comando de compilação ao compilar testes da pasta *`freertos-source`*`/demos`.

`clientWifiConfig`

Configuração do Wi-Fi do cliente. Os testes da biblioteca de Wi-Fi exigem que uma placa de MCU seja conectada a dois pontos de acesso. (Os dois pontos de acesso podem ser os mesmos.) Este atributo define as configurações de Wi-Fi para o primeiro ponto de acesso. Alguns dos casos de teste de Wi-Fi esperam que o ponto de acesso tenha algum recurso de segurança e que não esteja aberto. Verifique se os dois pontos de acesso estão na mesma sub-rede do computador host que executa o IDT.

`wifi_ssid`

O SSID do Wi-Fi.

`wifi_password`

A senha do Wi-Fi.

`wifiSecurityType`

O tipo de segurança do Wi-Fi utilizada. Um dos valores:

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

> **Note**
>
> Se a placa não for compatível com Wi-Fi, você ainda deverá incluir a seção `clientWifiConfig` no arquivo `device.json`, mas poderá omitir os valores desses recursos.

`testWifiConfig`

Configuração de teste do Wi-Fi. Os testes da biblioteca de Wi-Fi exigem que uma placa de MCU seja conectada a dois pontos de acesso. (Os dois pontos de acesso podem ser os mesmos.)

Esse atributo define a configuração do Wi-Fi para o segundo ponto de acesso. Alguns dos casos de teste de Wi-Fi esperam que o ponto de acesso tenha algum recurso de segurança e que não esteja aberto. Verifique se os dois pontos de acesso estão na mesma sub-rede do computador host que executa o IDT.

`wifiSSID`

O SSID do Wi-Fi.

`wifiPassword`

A senha do Wi-Fi.

`wifiSecurityType`

O tipo de segurança do Wi-Fi utilizada. Um dos valores:

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

> **Note**
>
> Se a placa não for compatível com Wi-Fi, você ainda deverá incluir a seção `testWifiConfig` no arquivo `device.json`, mas poderá omitir os valores desses recursos.

`echoServerCertificateConfiguration`

O espaço reservado configurável de geração de certificados do servidor echo para testes de SSL. Este campo é obrigatório.

`certificateGenerationMethod`

Especifica se o certificado do servidor é gerado automaticamente ou fornecido manualmente.

`customPath`

Se o `certificateGenerationMethod` for "personalizado", `certificatePath` e `privateKeyPath` são obrigatórios.

`certificatePath`

Especifica o caminho do arquivo do certificado do servidor.

`privateKeyPath`

Especifica o caminho do arquivo da chave privada.

`eccCurveFormat`

Especifica o formato de curva compatível com a placa. Obrigatório quando `PKCS11` estiver definido como "ecc" em `device.json`. Os valores válidos são "P224", "P256", "P384" ou "P521".

`echoServerConfiguration`

As portas configuráveis do servidor de eco para WiFi testes de soquetes seguros. Esse campo é opcional.

`securePortForSecureSocket`

A porta que é usada para configurar um servidor echo com TLS para o teste de soquetes seguros. O valor padrão é 33333. Verifique se a porta configurada não está bloqueada por um firewall ou pela rede corporativa.

`insecurePortForSecureSocket`

A porta usada para configurar o servidor echo sem TLS para o teste de soquetes seguros. O valor padrão usado no teste é 33334. Verifique se a porta configurada não está bloqueada por um firewall ou pela rede corporativa.

`insecurePortForWiFi`

A porta usada para configurar o servidor de eco sem TLS para WiFi teste. O valor padrão usado no teste é 33335. Verifique se a porta configurada não está bloqueada por um firewall ou pela rede corporativa.

`otaConfiguration`

A configuração OTA. [Opcional]

`otaFirmwareFilePath`

O caminho completo para a imagem OTA criada após a compilação. Por exemplo, `{{testData.sourcePath}}/`*`relative-path/to/ota/image/from/source/root`*.

`deviceFirmwareFileName`

O caminho completo do arquivo no dispositivo MCU onde o firmware OTA está localizado. Alguns dispositivos não usam esse campo, mas você ainda deve fornecer um valor.

`otaDemoConfigFilePath`

O caminho completo para `aws_demo_config.h`, encontrado em *`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/`. Esses arquivos estão incluídos no modelo de código de transferência fornecido pelo FreeRTOS.

`codeSigningConfiguration`

A configuração de assinatura de código.

`signingMethod`

O método de assinatura de código. Os valores possíveis são `AWS` ou `Custom`.

> **Note**
>
> Nas regiões de Pequim e Ningxia, use `Custom`. Não há compatibilidade para a assinatura de código da `AWS` nessas regiões.

`signerHashingAlgorithm`

O algoritmo de hash ao qual o dispositivo oferece suporte. Os valores possíveis são `SHA1` ou `SHA256`.

`signerSigningAlgorithm`

O algoritmo de assinatura ao qual o dispositivo oferece suporte. Os valores possíveis são `RSA` ou `ECDSA`.

`signerCertificate`

O certificado de confiança usado para OTA.

Para o método de assinatura de AWS código, use o Amazon Resource Name (ARN) para o certificado confiável enviado para o. AWS Certificate Manager

Para o método de assinatura de código personalizado, use o caminho absoluto para o arquivo de certificado do assinante.

Para obter mais informações sobre como criar um certificado confiável, consulte Criação de um certificado de assinatura de código.

`signerCertificateFileName`

O nome do arquivo do certificado de assinatura de código no dispositivo. Esse valor deve corresponder ao nome do arquivo que você forneceu ao executar o comando `aws acm import-certificate`.

Para ter mais informações, consulte Criação de um certificado de assinatura de código.

`compileSignerCertificate`

Defina como `true` se o certificado de verificação de assinatura do signatário do código não for provisionado ou atualizado, então ele deve ser compilado no projeto. AWS IoT Device Tester busca o certificado confiável e o compila em. `aws_codesigner_certifiate.h`

`untrustedSignerCertificate`

O ARN ou caminho de arquivo de um segundo certificado usado em alguns testes OTA como um certificado não confiável. Para obter mais informações sobre como criar um certificado, consulte Criar um certificado de assinatura de código.

`signerPlatform`

O algoritmo de assinatura e hashing que o AWS Code Signer usa ao criar o trabalho de atualização do OTA. Atualmente, os valores possíveis para este campo são `AmazonFreeRTOS-TI-CC3220SF` e `AmazonFreeRTOS-Default`.

- Escolha `AmazonFreeRTOS-TI-CC3220SF` se `SHA1` e `RSA`.
- Escolha `AmazonFreeRTOS-Default` se `SHA256` e `ECDSA`.

Se você precisar de `SHA256` | `RSA` ou `SHA1` | `ECDSA` para sua configuração, entre em contato conosco para obter mais suporte.

Configure `signCommand` se você escolheu `Custom` para `signingMethod`.

`signCommand`

O comando usado para executar a assinatura de código personalizado. É possível encontrar o modelo no diretório `/configs/script_templates`.

Dois espaços reservados `{{inputImageFilePath}}` e `{{outputSignatureFilePath}}` são necessários no comando. `{{inputImageFilePath}}` é o caminho do arquivo da imagem criada pelo IDT a ser

assinado. `{{outputSignatureFilePath}}` é o caminho do arquivo da assinatura que será gerado pelo script.

`cmakeConfiguration`

Configuração CMake [Opcional]

> **Note**
>
> Para executar casos de teste CMake, você deve fornecer o nome da placa, nome do fornecedor e o `frToolchainPath` ou `compilerName`. Você também pode fornecer o `cmakeToolchainPath` se você tiver um caminho personalizado para a cadeia de ferramentas do CMake.

`boardName`

O nome da placa em teste. O nome do painel deve ser o mesmo que o nome da pasta em *`path/to/afr/source/code`*`/vendors/`*`vendor`*`/boards/`*`board`*.

`vendorName`

O nome do fornecedor para a placa em teste. O fornecedor deve ser o mesmo que o nome da pasta em *`path/to/afr/source/code`*`/vendors/`*`vendor`*.

`compilerName`

O nome do compilador.

`frToolchainPath`

O caminho totalmente qualificado para o conjunto de ferramentas do compilador

`cmakeToolchainPath`

O caminho totalmente qualificado para o conjunto de ferramentas CMake. Este campo é opcional

`freertosFileConfiguration`

A configuração dos arquivos FreeRTOS que o IDT modifica antes de executar testes.

`required`

Esta seção especifica os testes necessários dos arquivos de configuração movidos, por exemplo, PKCS11, TLS e assim por diante.

`configName`

O nome do teste que é configurado.

`filePath`

O caminho absoluto para os arquivos de configuração no repositório *`freertos`*. Use a variável `{{testData.sourcePath}}` para definir o caminho.

`optional`

Esta seção especifica testes opcionais cujos arquivos de configuração você moveu, por exemplo, OTA WiFi, etc.

`configName`

O nome do teste que é configurado.

`filePath`

O caminho absoluto para os arquivos de configuração no repositório *`freertos`*. Use a variável `{{testData.sourcePath}}` para definir o caminho.

> **Note**
>
> Para executar casos de teste CMake, você deve fornecer o nome da placa, nome do fornecedor e o `afrToolchainPath` ou `compilerName`. Você também pode fornecer o `cmakeToolchainPath` se você tiver um caminho personalizado para o conjunto de ferramentas CMake.

Variáveis do IDT para FreeRTOS

Os comandos para criar seu código e atualizar o dispositivo podem exigir conectividade ou outras informações sobre seus dispositivos para serem executados com êxito. AWS IoT Device Tester permite que você faça referência às informações do dispositivo em flash e crie comandos usando JsonPatho. Usando JsonPath expressões simples, você pode buscar as informações necessárias especificadas em seu `device.json` arquivo.

Variáveis de caminho

O IDT para FreeRTOS define as seguintes variáveis de caminho que podem ser usadas em linhas de comando e arquivos de configuração:

`{{testData.sourcePath}}`

Expande o caminho do código-fonte. Se você usar essa variável, ela deverá ser usada nos comandos de compilação e atualização.

`{{sdkPath}}`

Expande-se para o valor em seu `userData.sdkConfiguration.path` quando usado nos comandos compilar e atualizar.

`{{device.connectivity.serialPort}}`

Expande à porta serial.

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

Expande o número de série do dispositivo.

`{{enableTests}}`

Valor inteiro que indica se a compilação é para testes (valor 1) ou demonstrações (valor 0).

`{{buildImageName}}`

O nome do arquivo da imagem criada pelo comando de compilação.

`{{otaCodeSignerPemFile}}`

Arquivo PEM para o assinante do código OTA.

`{{config.idtRootPath}}`

Expande-se até o caminho AWS IoT Device Tester raiz. Essa variável substitui o caminho absoluto do IDT quando usada pelos comandos build e flash.

Use a interface de usuário do IDT para FreeRTOS para executar o pacote de qualificação do FreeRTOS

A partir do IDT v4.3.0, para AWS IoT Device Tester FreeRTOS (IDT-FreeRTOS) inclui uma interface de usuário baseada na web que permite interagir com o executável da linha de comando do IDT e os arquivos de configuração relacionados. Você pode usar a interface do IDT-FreeRTOS para criar uma nova configuração para executar testes de IDT ou modificar uma configuração existente. Também é possível usar a interface do usuário para invocar o executável do IDT e executar testes.

A interface de usuário do IDT-FreeRTOS fornece as seguintes funções:

- Simplifique a configuração dos arquivos de configuração para testes do IDT-FreeRTOS.
- Simplifique o uso do IDT-FreeRTOS para executar testes de qualificação.

Para obter informações sobre o uso da linha de comando para executar os testes de qualificação, consulte Preparação para testar sua placa de microcontrolador pela primeira vez.

Esta seção descreve os pré-requisitos da interface do usuário do IDT para FreeRTOS e mostra como começar a executar testes de qualificação da interface do usuário.

Tópicos

Pré-requisitos

Esta seção descreve os pré-requisitos para testar microcontroladores com o AWS IoT Device Tester.

Tópicos

Usar um navegador da web compatível

A interface de usuário do IDT para FreeRTOS é compatível com os seguintes navegadores da web.

Navegador	Version (Versão)
Google Chrome	Últimas três versões principais
Mozilla Firefox	Últimas três versões principais

Navegador	Version (Versão)
Microsoft Edge	Últimas três versões principais
Apple Safari para macOS	Últimas três versões principais

Recomendamos que você use o Google Chrome ou o Mozilla Firefox para uma ter melhor experiência.

> **Note**
>
> A interface de usuário do IDT para FreeRTOS não é compatível com o Microsoft Internet Explorer.

Faça download do FreeRTOS

Você pode baixar uma versão do FreeRTOS com o seguinte GitHubcomando:

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

em que <FREERTOS_RELEASE_VERSION> é uma versão do FreeRTOS (por exemplo, 202007.00) correspondente a uma versão do IDT listada em Versões compatíveis do AWS IoT Device Tester para FreeRTOS. Isso garante que você tenha o código-fonte completo, incluindo submódulos, e esteja usando a versão correta do IDT para sua versão de FreeRTOS e vice-versa.

O Windows tem uma limitação de comprimento de caminho de 260 caracteres. A estrutura de caminho do FreeRTOS tem muitos níveis; portanto, se você estiver usando o Windows, mantenha os caminhos de arquivo dentro do limite de 260 caracteres. Por exemplo, clone o FreeRTOS para `C:\FreeRTOS` em vez de `C:\Users\username\programs\projects\myproj\FreeRTOS\`.

Considerações para a qualificação LTS (qualificação para FreeRTOS que usa bibliotecas LTS)

- Para que seu microcontrolador seja designado como compatível com versões de FreeRTOS baseadas em suporte de longo prazo (LTS) no AWS Partner Device Catalog, você deve fornecer

um arquivo manifesto. Para obter mais informações, consulte a Lista de verificação de qualificação do FreeRTOS no Guia de qualificação do FreeRTOS.

- Para validar se seu microcontrolador suporta versões baseadas em LTS do FreeRTOS e qualificá-lo para envio ao AWS Partner Device Catalog, você deve usar (AWS IoT Device Tester IDT) com a suíte de testes FreeRTOS Qualification (FRQ) versão v1.4.x.
- O suporte para versões baseadas em LTS do FreeRTOS está limitado à versão 202012.xx do FreeRTOS.

Faça download do IDT para FreeRTOS

Cada versão do FreeRTOS tem uma versão correspondente do IDT para o FreeRTOS para realizar testes de qualificação. Baixe a versão adequada do IDT para FreeRTOS em Versões compatíveis do AWS IoT Device Tester para FreeRTOS.

Extraia o IDT para FreeRTOS em um local no sistema de arquivos onde existam permissões de leitura e gravação. Como o Microsoft Windows tem um limite de caracteres para o comprimento do caminho, extraia o IDT para FreeRTOS em um diretório raiz, como `C:\` ou `D:\`.

> **Note**
>
> Recomendamos que você extraia o pacote IDT em uma unidade local. Permitir que vários usuários executem o IDT em um local compartilhado, como um diretório NFS ou uma pasta compartilhada na rede do Windows, pode resultar na falta de resposta do sistema ou na corrupção de dados.

Crie e configure uma AWS conta

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

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

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

AWS IoT Device Tester política gerenciada

Para habilitar o Device Tester a executar e coletar métricas, a política gerenciada `AWSIoTDeviceTesterForFreeRTOSFullAccess` contém as seguintes permissões:

- `iot-device-tester:SupportedVersion`

  Concede permissão para obter a lista de versões do FreeRTOS e versões do pacote de testes compatíveis com o IDT, para que elas estejam disponíveis na AWS CLI.
- `iot-device-tester:LatestIdt`

  Concede permissão para obter a AWS IoT Device Tester versão mais recente que está disponível para download.
- `iot-device-tester:CheckVersion`

  Concede permissão para verificar se uma combinação de produtos, conjunto de testes e versões do AWS IoT Device Tester são compatíveis.
- `iot-device-tester:DownloadTestSuite`

  Concede permissão AWS IoT Device Tester para baixar suítes de teste.

- `iot-device-tester:SendMetrics`

  Concede permissão para publicar dados AWS IoT Device Tester de métricas de uso.

Conceitos básicos da interface do usuário do IDT-FreeRTOS

Esta seção mostra como usar a interface de usuário do IDT-FreeRTOS para criar ou modificar sua configuração e, em seguida, mostra como executar testes.

Tópicos

- Configurar AWS credenciais
- Abra a interface de usuário do IDT-FreeRTOS
- Criar uma configuração
- Modificar uma configuração existente
- Execute testes de qualificação

Configurar AWS credenciais

Você deve configurar as credenciais para o AWS usuário que você criou emCrie e configure uma AWS conta. Você pode especificar suas credenciais de uma das seguintes formas:

- Em um arquivo de credenciais
- Como variáveis de ambiente

Configurar AWS credenciais com um arquivo de credenciais

O IDT usa o mesmo arquivo de credenciais que a AWS CLI. Para obter mais informações, consulte Arquivos de configuração e credenciais.

O local do arquivo de credenciais varia de acordo com o sistema operacional que você está usando:

- macOS, Linux: `~/.aws/credentials`
- Windows: `C:\Users\UserName\.aws\credentials`

Adicione suas AWS credenciais ao `credentials` arquivo no seguinte formato:

```
[default]
```

```
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

> **Note**
>
> Se você não usar o `default` AWS perfil, certifique-se de especificar o nome do perfil na interface do IDT-FreeRTOS. Para obter mais informações sobre perfis, consulte Perfis nomeados.

Configurar AWS credenciais com variáveis de ambiente

As variáveis de ambiente são variáveis mantidas pelo sistema operacional e usadas pelos comandos do sistema. Elas não são salvas se você fechar a sessão SSH. A interface de usuário do IDT-FreeRTOS usa as variáveis de ambiente `AWS_ACCESS_KEY_ID` e `AWS_SECRET_ACCESS_KEY` para armazenar suas credenciais da AWS .

Para definir essas variáveis no Linux, macOS ou Unix, use :export

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Para definir essas variáveis no Windows, use :set

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Abra a interface de usuário do IDT-FreeRTOS

Como abrir a interface de usuário do IDT-FreeRTOS

1. Baixe uma versão compatível do IDT-FreeRTOS e extraia o arquivo baixado em um local em seu sistema de arquivos onde existam permissões de leitura e gravação.
2. Execute o comando a seguir para navegar até o diretório de instalação do IDT-FreeRTOS:

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

3. Execute o comando a seguir para abrir a interface de usuário do IDT-FreeRTOS:

Linux

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

Windows

```
./devicetestergui_win_x64-64
```

macOS

```
./devicetestergui_mac_x86-64
```

> **Note**
>
> No Mac, para permitir que seu sistema execute a interface do usuário, acesse Preferências do sistema -> Segurança e privacidade. Ao executar os testes, é aconselhável repeti-los mais três vezes.

A interface de usuário do IDT-FreeRTOS é aberta em seu navegador padrão. Para obter informações sobre os navegadores compatíveis, consulte Usar um navegador da web compatível.

Criar uma configuração

Se for um usuário iniciante, deverá criar uma nova configuração para definir os arquivos de configuração JSON que o IDT-FreeRTOS exige para executar testes. Em seguida, você pode executar testes ou modificar a configuração criada.

Para obter exemplos dos arquivos `config.json`, `device.json` e `userdata.json`, consulte Preparação para testar sua placa de microcontrolador pela primeira vez. Para obter um exemplo do arquivo `resource.json` usado somente para executar testes de Bluetooth Low Energy (BLE), consulte Execução de testes de Bluetooth Low Energy.

Como criar uma configuração

1. Na interface de usuário do IDT-FreeRTOS, abra o menu de navegação e escolha Criar configuração.

   > **Important**
   >
   > Você deve configurar suas AWS credenciais antes de abrir a interface do usuário. Se não configurou suas credenciais, feche a janela do navegador da interface de usuário do IDT-FreeRTOS, siga as etapas em Configurar AWS credenciais e reabra a interface de usuário do IDT-FreeRTOS.

2. Siga o assistente de configuração para inserir as configurações do IDT que são usadas para executar os testes de qualificação. O assistente define as seguintes configurações nos arquivos de configuração JSON localizados no diretório *devicetester-extract-location*`/config`.

   - AWS settings — As Conta da AWS informações que o IDT-FreeRTOS usa para criar AWS recursos durante as execuções de teste. Essas configurações são definidas no arquivo `config.json`.
   - Repositório FreeRTOS: o caminho absoluto para o repositório e o código portado do FreeRTOS, e o tipo de qualificação que você deseja realizar. Essas configurações são definidas no arquivo `userdata.json`.

     Você deve transferir o FreeRTOS para o seu dispositivo antes de executar os testes de qualificação. Para obter mais informações, consulte o Guia de portabilidade do FreeRTOS
   - Compilar e atualizar: os comandos de compilação e atualização para seu hardware que permitem que o IDT compile e instale testes em sua placa automaticamente. Essas configurações são definidas no arquivo `userdata.json`.
   - Dispositivos: as configurações do grupo de dispositivos para os dispositivos a serem testados. Essas configurações são definidas nos campos `id` e `sku` no bloco `devices` do grupo de dispositivos no arquivo `device.json`.
   - Rede: as configurações para testar o suporte de comunicação de rede para seus dispositivos. Essas configurações são definidas no bloco `features` do arquivo `device.json` e nos blocos `clientWifiConfig` e `testWifiConfig` do arquivo `userdata.json`.
   - Servidor echo: as configurações do servidor echo para testes de soquete seguros. Essas configurações são definidas no arquivo `userdata.json`.

     Para obter mais informações sobre a configuração do servidor echo, consulte https://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html.

- CMake: (opcional) as configurações para executar os testes de funcionalidade de compilação do CMake. Essa configuração é necessária somente se você estiver usando o CMake como sistema de compilação. Essas configurações são definidas no arquivo `userdata.json`.
- BLE: as configurações para executar testes de funcionalidade Bluetooth Low Energy. Essas configurações são definidas no bloco `features` do arquivo `device.json` e no arquivo `resource.json`.
- OTA: as configurações para executar testes de funcionalidade OTA. Essas configurações são definidas no bloco `features` do arquivo `device.json` e no arquivo `userdata.json`.

3. Na página Revisar, verifique suas informações de configuração.

Depois de concluir a revisão da configuração, para executar os testes de qualificação, escolha Executar testes.

Modificar uma configuração existente

Se já configurou os arquivos de configuração para o IDT, pode usar a interface de usuário do IDT-FreeRTOS para modificar a configuração existente. Verifique se os arquivos de configuração existentes estão disponíveis no diretório *`devicetester-extract-location`*`/config`.

Como modificar uma configuração nova

1. Na interface de usuário do IDT-FreeRTOS, abra o menu de navegação e escolha Editar configuração existente.

   O painel de configuração exibe informações sobre suas configurações existentes. Se uma configuração estiver incorreta ou indisponível, o status dessa configuração será `Error validating configuration`.
2. Para modificar uma configuração existente, conclua estas etapas:

   a. Escolha o nome de uma configuração para abrir sua página de ajustes.

   b. Modifique as configurações e escolha Salvar para regenerar o arquivo de configuração correspondente.

Depois de concluir a modificação da configuração, verifique se todas as configurações passaram pela validação. Se o status de cada configuração for `Valid`, será possível executar seus testes de qualificação usando essa configuração.

Execute testes de qualificação

Depois de criar uma configuração para o IDT-FreeRTOS, é possível executar seus testes de qualificação.

Como executar testes de qualificação

1. Para validar a configuração.
2. No menu de navegação, escolha Executar testes.
3. Escolha Iniciar testes para iniciar a execução do teste.

O IDT-FreeRTOS executa os testes de qualificação e exibe o resumo da execução do teste e quaisquer erros no console do Executor de testes. Depois que a execução do teste for concluída, você poderá visualizar os resultados e os logs do teste nos seguintes locais:

- Os resultados dos testes estão localizados no diretório *devicetester-extract-location*/`results/`*execution-id*.
- Os logs de teste estão localizados no diretório *devicetester-extract-location*/`results/`*execution-id*`/logs`.

Para obter mais informações sobre os resultados de teste e logs, consulte Noções básicas de resultados e logs.

Execução de testes de Bluetooth Low Energy

Esta seção descreve como configurar e executar os testes de Bluetooth usando os AWS IoT Device Tester Freertos. Os testes Bluetooth não são necessários para a qualificação principal. Se você não deseja testar seu dispositivo com o suporte a Bluetooth do FreeRTOS, pode ignorar esta configuração. Deixe o recurso BLE no device.json definido como `No`.

Pré-requisitos

- Siga as instruções em Preparação para testar sua placa de microcontrolador pela primeira vez.
- Um Raspberry Pi 4B ou 3B+. (Obrigatório para executar a aplicação complementar Raspberry Pi BLE)
- Um micro cartão SD e adaptador de cartão SD para o software Raspberry Pi.

Configuração do Raspberry Pi

Para testar os recursos de BLE do dispositivo em teste (DUT), você deve ter um Raspberry Pi Modelo 4B ou 3B+.

Para configurar o Raspberry Pi para executar testes de BLE

1. Baixe uma das imagens Yocto personalizadas que contenham o software necessário para executar os testes.

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

   > **Note**
   >
   > A imagem do Yocto só deve ser usada para testes com AWS IoT Device Tester FreeRTOS e não para qualquer outra finalidade.

2. Atualize a imagem yocto no cartão SD do Raspberry Pi.

   - Usando uma ferramenta de gravação de cartão SD, como Etcher, instale o arquivo *image-name*`.rpi-sd.img` obtido por download no cartão SD. Como a imagem do sistema operacional é grande, essa etapa deve levar algum tempo. Depois ejete seu cartão SD do computador e insira o cartão microSD no seu Raspberry Pi.

3. Configure seu Raspberry Pi.

   a. Durante a primeira inicialização, recomendamos que você conecte o Raspberry Pi a um monitor, teclado e mouse.

   b. Conecte seu Raspberry Pi a uma fonte de alimentação micro USB.

   c. Faça login usando as credenciais padrão. Para ID de usuário, insira **`root`**. Para senha, insira **`idtafr`**.

   d. Usando uma conexão Wi-Fi ou Ethernet, conecte o Raspberry Pi à sua rede.

      i. Para conectar seu Raspberry Pi por Wi-Fi, abra `/etc/wpa_supplicant.conf` no Raspberry Pi e adicione suas credenciais Wi-Fi à configuração 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. Execute `ifup wlan0` para iniciar a conexão Wi-Fi. Pode levar alguns minutos para se conectar à sua rede Wi-Fi.

e. Para uma conexão Ethernet, execute `ifconfig eth0`. Para uma conexão Wi-Fi, execute `ifconfig wlan0`. Anote o endereço IP, que aparece como `inet addr` na saída do comando. Você precisa do endereço IP posteriormente neste procedimento.

f. (Opcional) Os testes executam comandos no Raspberry Pi sobre SSH usando as credenciais padrão para a imagem yocto. Para segurança adicional, recomendamos que você configure a autenticação de chave pública para SSH e desative o SSH baseado em senha.

i. Crie uma chave SSH usando o comando OpenSSL `ssh-keygen`. Se você já tem um par de chaves SSK em seu computador host, é uma prática recomendada criar um novo AWS IoT Device Tester para permitir que os FreeRTOS façam login no seu Raspberry Pi.

> **Note**
>
> O Windows não vem com um cliente SSH instalado. Para obter informações sobre como instalar um cliente SSH no Windows, consulte Como fazer download do software SSH.

ii. O comando `ssh-keygen` solicita que você informe um nome e caminho para armazenar o par de chaves. Por padrão, os arquivos de pares de chaves são nomeados como `id_rsa` (chave privada) e `id_rsa.pub` (chave pública). No macOS e no Linux, o local padrão desses arquivos é `~/.ssh/`. No Windows, o local padrão é `C:\Users\user-name`.

iii. Quando uma frase-chave for solicitada, bastará pressionar ENTER para continuar.

iv. Para adicionar sua chave SSH ao Raspberry Pi para que os AWS IoT Device Tester FreeRTOS possam entrar no dispositivo, use o comando do seu computador host. `ssh-copy-id` Esse comando adiciona sua chave pública ao arquivo `~/.ssh/authorized_keys` no seu Raspberry Pi.

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

v. Quando solicitado a inserir uma senha, digite **`idtafr`**. Essa é a senha padrão para a imagem yocto.

> **Note**
>
> O comando `ssh-copy-id` pressupõe que a chave pública é chamada `id_rsa.pub`. No macOS e Linux, o local padrão é `~/.ssh/`. No Windows, o local padrão é `C:\Users\user-name\.ssh`. Se você atribuiu à chave pública um nome diferente ou armazenou em um local diferente, você deve especificar o caminho para sua chave pública SSH usando a opção `-i` para `ssh-copy-id` (por exemplo, `ssh-copy-id -i ~/my/path/myKey.pub`). Para obter mais informações sobre a criação de chaves SSH e a cópia de chaves públicas, consulte SSH-COPY-ID.

vi. Para testar se a autenticação da chave pública está funcionando, execute `ssh -i /my/path/myKey root@raspberry-pi-device-ip`.

Se você não receber uma solicitação de senha, a autenticação de chave pública está funcionando.

vii. Verifique se é possível fazer login no seu Raspberry Pi usando uma chave pública e, depois, desative o SSH baseado em senha.

A. No Raspberry Pi, edite o arquivo `/etc/ssh/sshd_config`.

B. Defina o recurso `PasswordAuthentication` como no.

C. Salve e feche o arquivo `sshd_config`.

D. Recarregue o servidor SSH executando `/etc/init.d/sshd reload`.

g. Crie um arquivo `resource.json`.

i. No diretório em que você extraiu o AWS IoT Device Tester, crie um arquivo chamado. `resource.json`

ii. Adicione as seguintes informações sobre seu Raspberry Pi ao arquivo, *rasp-pi-ip-address*substituindo-as pelo endereço IP do seu 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. Se você decidir não usar a autenticação de chave pública para SSH, adicione o seguinte à seção `connectivity` do arquivo `resource.json`.

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

iv. (Opcional) Se você decidir usar a autenticação de chave pública para SSH, adicione o seguinte à seção `connectivity` do arquivo `resource.json`.

```
"connectivity": {
    "protocol": "ssh",
    "ip": "rasp-pi-ip-address",
```

```
    "auth": {
        "method": "pki",
        "credentials": {
            "user": "root",
            "privKeyPath": "location-of-private-key"
        }
    }
}
```

Configuração do dispositivo do FreeRTOS

No seu arquivo `device.json`, defina o recurso `BLE` como `Yes`. Se você estiver começando com um arquivo `device.json` de antes de os testes de Bluetooth estarem disponíveis, é preciso adicionar o recurso para o BLE à matriz `features`:

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

Execução de testes de BLE

Depois de habilitar o recurso BLE no `device.json`, os testes BLE serão executados quando você executar `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite` sem especificar um group-id.

Se você quiser executar os testes BLE separadamente, poderá especificar o ID do grupo para o BLE: `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite --userdata` *`path-to-userdata`*`/userdata.json --group-id FullBLE`.

Para o desempenho mais confiável, coloque o Raspberry Pi próximo do dispositivo em teste (DUT).

Solução de problemas de testes de BLE

Siga as etapas em Preparação para testar sua placa de microcontrolador pela primeira vez. Se testes diferentes de BLE estiverem falhando, é mais provável que o problema não seja decorrente da configuração do Bluetooth.

Como executar o pacote de qualificação do FreeRTOS

Você usa o executável AWS IoT Device Tester for FreeRTOS para interagir com o IDT for FreeRTOS. Os exemplos de linha de comando a seguir mostram como executar os testes de qualificação para um grupo de dispositivos (um conjunto de dispositivos idênticos).

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
```

Executa um conjunto de testes em um grupo de dispositivos. O arquivo `userdata.json` deve estar localizado no diretório *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/`.

> **Note**
>
> Se estiver executando o IDT para FreeRTOS no Windows, use barras (/) para especificar o caminho até o arquivo `userdata.json`.

Use o seguinte comando para executar um grupo de testes específico:

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

```
    --userdata userdata.json
```

Os parâmetros `suite-id` e `pool-id` são opcionais se você estiver executando um único conjunto de testes em um único grupo de dispositivos (ou seja, você tem apenas um grupo de dispositivos definido no arquivo `device.json`).

Use o seguinte comando para executar um caso de teste em um grupo de testes:

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

Você pode usar o comando `list-test-cases` para listar os casos de teste em um grupo de testes.

Opções de linhas de comando do IDT para FreeRTOS

group-id

(Opcional) Os grupos de testes a serem executados, como uma lista separada por vírgulas. Se não for especificado, o IDT executará todos os grupos de testes no conjunto de testes.

pool-id

(Opcional) O grupo de dispositivos a ser testado. Isso será necessário se você definir vários grupos de dispositivos no `device.json`. Se você tiver apenas um grupo de dispositivos, poderá omitir essa opção.

suite-id

(Opcional) A versão do conjunto de testes a ser executada. Se não for especificada, o IDT usará a versão mais recente no diretório de testes em seu sistema.

> **Note**
>
> Começando com o IDT v3.0.0, o IDT verifica se há conjuntos de testes mais recentes online. Para ter mais informações, consulte Versões do conjunto de testes.

test-id

(Opcional) Os testes a serem executados, como uma lista separada por vírgulas. Se especificado, `group-id` deve especificar um único grupo.

Example

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

update-idt

(Opcional) Se este parâmetro não estiver definido e houver uma versão mais recente do IDT disponível, será solicitado atualizar o IDT. Se esse parâmetro for definido como Y, o IDT interromperá a execução do teste se detectar que uma versão mais recente está disponível. Se esse parâmetro for definido como N, o IDT continuará a execução do teste.

update-managed-policy

(Opcional) Se esse parâmetro não for usado e o IDT detectar que sua política gerenciada não é up-to-date, você será solicitado a atualizar sua política gerenciada. Se esse parâmetro for definido comoY, o IDT interromperá a execução do teste se detectar que sua política gerenciada não está. up-to-date Se esse parâmetro for definido como N, o IDT continuará a execução do teste.

upgrade-test-suite

(Opcional) Se não for usado e uma versão mais recente do conjunto de testes estiver disponível, você será solicitado a fazer download. Para ocultar o prompt, especifique y para sempre fazer download do conjunto de testes mais recente, ou n para usar o conjunto de testes especificado ou a versão mais recente em seu sistema.

Example

Exemplo

Para sempre fazer download e usar o conjunto de testes mais recente, use o seguinte comando.

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

Para usar o conjunto de testes mais recente em seu sistema, use o seguinte comando.

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

h

Use a opção de ajuda para saber mais sobre as opções de `run-suite`.

Example

Exemplo

```
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
```

O arquivo `userdata.json` deve estar localizado no diretório *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/`.

> **Note**
>
> Se você estiver executando o IDT para FreeRTOS no Windows, use barras (/) para especificar o caminho até o arquivo `userdata.json`.

Use o seguinte comando para executar um grupo de testes específico.

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

`suite-id` e `pool-id` são opcionais se você estiver executando um único conjunto de testes em um único grupo de dispositivos (ou seja, você tem apenas um grupo de dispositivos definido no arquivo `device.json`).

Opções de linhas de comando do IDT para FreeRTOS

group-id

(Opcional) Especifica o grupo de testes.

pool-id

Especifica o grupo de dispositivos para testar. Se você tiver apenas um grupo de dispositivos, poderá omitir essa opção.

suite-id

(Opcional) Especifica o conjunto de testes a ser executado.

Comandos do IDT para FreeRTOS

O comando do IDT para FreeRTOS é compatível com as seguintes operações:

IDT v3.0.0 and later

`help`

Lista as informações sobre o comando especificado.

`list-groups`

Lista os grupos em um determinado conjunto.

`list-suites`

Lista os conjuntos disponíveis.

`list-supported-products`

Lista os produtos compatíveis e as versões do conjunto de testes.

`list-supported-versions`

Lista as versões do FreeRTOS e do pacote de teste compatíveis com a versão atual do IDT.

`list-test-cases`

Lista os casos de teste em um grupo especificado.

`run-suite`

Executa um conjunto de testes em um grupo de dispositivos.

Use a opção `--suite-id` para especificar uma versão do conjunto de testes, ou omiti-la para usar a versão mais recente em seu sistema.

Use o `--test-id` para executar um caso de teste individual.

Example

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

Para obter uma lista completa de opções, consulte Como executar o pacote de qualificação do FreeRTOS.

> **Note**
>
> Começando com o IDT v3.0.0, o IDT verifica se há conjuntos de testes mais recentes online. Para ter mais informações, consulte Versões do conjunto de testes.

IDT v1.7.0 and earlier

`help`

Lista as informações sobre o comando especificado.

`list-groups`

Lista os grupos em um determinado conjunto.

`list-suites`

Lista os conjuntos disponíveis.

`run-suite`

Executa um conjunto de testes em um grupo de dispositivos.

Teste para requalificação

À medida que novas versões de testes de qualificação do IDT para FreeRTOS são lançadas, ou ao atualizar os pacotes ou drivers de dispositivo específicos da placa, você pode usar o IDT para FreeRTOS para testar as placas do microcontrolador. Para qualificações subsequentes, certifique-se

de que tem as versões mais recentes do FreeRTOS e do IDT para FreeRTOS e execute os testes de qualificação novamente.

Noções básicas de resultados e logs

Esta seção descreve como visualizar e interpretar os resultados de relatórios e logs do IDT.

Visualização de resultados

Enquanto em execução, o IDT grava erros no console, arquivos de log e relatórios de teste. Depois de concluir o pacote de teste de qualificação, O IDT gravará um resumo da execução de teste no console e gerará dois relatórios de teste. Esses relatórios podem ser encontrados em *devicetester-extract-location*`/results/`*execution-id*`/`. Ambos os relatórios capturam os resultados da execução do pacote de teste de qualificação.

Esse `awsiotdevicetester_report.xml` é o relatório do teste de qualificação que você envia AWS para listar seu dispositivo no Catálogo de dispositivos do AWS parceiro. O relatório contém os seguintes elementos:

- A versão do IDT para FreeRTOS.
- A versão do FreeRTOS que foi testada.
- Os recursos do FreeRTOS que são compatíveis com o dispositivo com base nos testes aprovados.
- A SKU e o nome do dispositivo especificados no arquivo `device.json`.
- Os recursos do dispositivo especificados no arquivo `device.json`.
- O resumo agregado dos resultados de casos de teste.
- Um detalhamento dos resultados dos casos de teste por bibliotecas que foram testadas com base nos recursos do dispositivo (por exemplo FullWiFi, FullMQTT e assim por diante).
- Caso essa qualificação do FreeRTOS seja para a versão 202012.00 que usa bibliotecas LTS.

O `FRQ_Report.xml` é um relatório no formato padrão JUnit XML. É possível integrá-lo a plataformas de CI/CD, como Jenkins, Bamboo e assim por diante. O relatório contém os seguintes elementos:

- Um resumo agregado dos resultados de casos de teste.
- Um detalhamento dos resultados de casos de teste por bibliotecas que foram testadas com base nos recursos do dispositivo.

Como interpretar os resultados do IDT para FreeRTOS

A seção de relatório em `awsiotdevicetester_report.xml` ou em `FRQ_Report.xml` lista os resultados dos testes que são executados.

A primeira tag XML `<testsuites>` contém o resumo geral da execução do teste. Por exemplo: .

```
<testsuites name="FRQ results" time="5633" tests="184" failures="0"
errors="0" disabled="0">
```

recursos usados na tag **`<testsuites>`**

`name`

O nome do conjunto de testes.

`time`

O tempo, em segundos, necessário para executar o conjunto de qualificação.

`tests`

O número de casos de teste executados.

`failures`

O número de casos de teste que foram executados, mas não foram aprovados.

`errors`

O número de casos de teste que o IDT para FreeRTOS não pôde executar.

`disabled`

Esse recurso não é usado e pode ser ignorado.

Se não houver falhas ou erros no caso de teste, seu dispositivo atende aos requisitos técnicos para executar Freertos e pode interoperar com os serviços. AWS IoT Se você optar por listar seu dispositivo no Catálogo de dispositivos do AWS parceiro, poderá usar esse relatório como evidência de qualificação.

Se houver falhas de caso de teste ou erros, você poderá identificar o caso de teste com falha analisando as tags XML `<testsuites>`. As tags XML `<testsuite>` dentro da tag `<testsuites>` mostram o resumo do resultado do caso de teste para um grupo de testes.

```
<testsuite name="FullMQTT" package="" tests="16" failures="0" time="76"
disabled="0" errors="0" skipped="0">
```

O formato é semelhante à tag `<testsuites>`, mas com um recurso chamado `skipped` que não é usado e pode ser ignorado. Dentro de cada tag XML `<testsuite>`, há tags `<testcase>` para cada um dos casos de teste que foram executados para um grupo de testes. Por exemplo: .

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase"
attempts="1"></testcase>
```

recursos usados na tag **`<awsproduct>`**

`name`

O nome do produto testado.

`version`

A versão do produto testado.

`sdk`

Se você executou o IDT com um SDK, esse bloco contém o nome e a versão do seu SDK. Se você não executou o IDT com um SDK, esse bloco contém:

```
<sdk>
    <name>N/A</vame>
    <version>N/A</version>
</sdk>
```

`features`

Os recursos validados. recursos marcados como `required` são necessários para enviar sua placa para qualificação. O snippet a seguir mostra como essas informações aparecem no arquivo `awsiotdevicetester_report.xml`.

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

Os recursos marcados como `optional` não são necessários para qualificação. Os seguintes trechos mostram recursos opcionais.

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

Se não há falhas de teste ou erros nos recursos exigidos, seu dispositivo atende aos requisitos técnicos para executar o FreeRTOS e pode interoperar com serviços do AWS IoT . Se quiser listar o dispositivo no AWS Partner Device Catalog, poderá usar este relatório como evidência de qualificação.

Se houver falhas de teste ou erros, você poderá identificar o teste com falha analisando as tags XML `<testsuites>`. As tags XML `<testsuite>` dentro da tag `<testsuites>` mostram o resumo do resultado do teste para um grupo de testes. Por exemplo: .

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

O formato é semelhante à tag `<testsuites>`, mas tem um recurso `skipped` que não é usado e pode ser ignorado. Dentro de cada tag XML `<testsuite>`, há tags `<testcase>` para cada teste executado para um grupo de testes. Por exemplo: .

```
<testcase classname="FreeRTOSVersion" name="FreeRTOSVersion"></testcase>
```

`lts`

Verdadeiro se a qualificação for para uma versão do FreeRTOS que usa bibliotecas LTS; caso contrário, falso.

recursos usados na tag **`<testcase>`**

`name`

O nome do caso de teste.

`attempts`

O número de vezes que o IDT para FreeRTOS executou o caso de teste.

Quando um teste falha ou ocorre um erro, as tags `<failure>` ou `<error>` são adicionadas à tag `<testcase>` com informações para a solução de problemas. Por exemplo: .

```
<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 ter mais informações, consulte Solução de problemas.

Visualizar logs do

É possível encontrar os logs que o IDT para FreeRTOS gera ao executar testes em *devicetester-extract-location*/results/*execution-id*/logs. Dois conjuntos de logs são gerados:

test_manager.log

Contém logs gerados de IDT para FreeRTOS (por exemplo, logs relacionados à configuração e geração de relatórios).

***test_group_id*__*test_case_id*.log** (por exemplo: **FullMQTT__Full_MQTT.log**)

O arquivo de log para um caso de teste, incluindo a saída do dispositivo em teste. O arquivo de log é nomeado de acordo com o grupo de teste e com o caso de teste que foi executado.

Usar o IDT para desenvolver e executar os próprios pacotes de testes

Começando do IDT v4.0.0, o IDT para FreeRTOS combina uma configuração padronizada e um formato de resultado com um ambiente de conjunto de testes que permite desenvolver conjuntos de testes personalizados para seus dispositivos e software de dispositivos. É possível adicionar testes personalizados para sua própria validação interna ou fornecê-los aos seus clientes para verificação de dispositivo.

Use o IDT para desenvolver e executar pacotes de testes personalizados da seguinte forma:

Para desenvolver pacotes de teste personalizados

- Crie pacotes de teste com lógica de testes personalizada para o dispositivo que deseja testar.
- Forneça ao IDT os pacotes de testes personalizados para os executores de teste. Inclua informações sobre configurações específicas para seus pacotes de teste.

Para executar pacotes de testes personalizados

- Configure o dispositivo que deseja testar.
- Implemente as configurações conforme exigido pelos pacotes de testes que deseja usar.

- Use o IDT para executar seus pacotes de teste personalizados.
- Veja os resultados do teste e os logs de execução dos testes executados pelo IDT.

Baixe a versão mais recente do AWS IoT Device Tester para FreeRTOS

Baixe a versão mais recente do IDT e extraia o software em um local no seu sistema de arquivos onde você tenha permissões de leitura e gravação.

> **Note**
>
> O IDT não oferece suporte a execução por vários usuários em um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Recomendamos extrair o pacote do IDT para uma unidade local e executar o binário do IDT na estação de trabalho local.
>
> O Windows tem uma limitação de comprimento de caminho de 260 caracteres. Se você estiver usando o Windows, extraia o IDT para um diretório raiz como `C:\` ou `D:\` para manter os caminhos abaixo do limite de 260 caracteres.

Fluxo de trabalho de criação de pacotes de teste

Os pacotes de teste são compostos por três tipos de arquivos:

- Arquivos de configuração que fornecem ao IDT informações sobre como executar o pacote de teste.
- Os arquivos executáveis de teste que o IDT usa para executar casos de teste.
- Arquivos adicionais necessários para executar testes.

Conclua as etapas básicas a seguir para criar testes de IDT personalizados:

1. Crie arquivos de configuração para seu pacote de teste.
2. Crie executáveis de casos de teste que contenham a lógica de teste para seu pacote de teste.
3. Verifique e documente as informações de configuração necessárias para que os executores de teste executem o pacote de teste.
4. Verifique se o IDT pode executar seu pacote de teste e produzir resultados de teste conforme o esperado.

Para criar rapidamente uma amostra de pacote personalizado e executá-la, siga as instruções em Tutorial: compile e execute o pacote de teste de amostra do IDT.

Para começar a criar um pacote de teste personalizado em Python, consulte Tutorial: desenvolva um pacote de teste simples do IDT.

Tutorial: compile e execute o pacote de teste de amostra do IDT

O AWS IoT Device Tester download inclui o código-fonte de uma amostra de suíte de testes. Você pode concluir este tutorial para criar e executar a suíte de testes de amostra para entender como você pode usar o FreeRTOS AWS IoT Device Tester para executar suítes de testes personalizadas. Embora este tutorial use SSH, é útil aprender a usar AWS IoT Device Tester com dispositivos FreeRTOS.

Neste tutorial, você concluirá as seguintes etapas:

1. Compile o pacote de teste de amostra
2. Usar o IDT para executar o pacote de teste de amostra

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- Requisitos do computador host
  - Versão mais recente do AWS IoT Device Tester
  - Python 3.7 ou posterior

    Para verificar a versão do Python instalada em seu computador, execute o seguinte comando:

    ```
    python3 --version
    ```

    No Windows, se o uso deste comando retornar um erro, use `python --version`. Se o número da versão retornada for 3.7 ou superior, execute o seguinte comando em um terminal do Powershell para definir `python3` como um alias para seu comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

Se nenhuma informação sobre versão for retornada ou se o número da versão for menor do que 3.7, siga as instruções em Fazer download do Python para instalar o Python 3.7+. Para obter mais informações, consulte a Documentação do Python.

- urllib3

  Para verificar se `urllib3` está instalado corretamente, execute o seguinte comando:

  ```
  python3 -c 'import urllib3'
  ```

  Se o `urllib3` não estiver instalado, execute o comando a seguir para instalá-lo:

  ```
  python3 -m pip install urllib3
  ```

- Requisitos do dispositivo
  - Um dispositivo com sistema operacional Linux e uma conexão de rede com a mesma rede do seu computador host.

    É recomendado o uso de um Raspberry Pi com o sistema operacional Raspberry Pi. Certifique-se de configurar o SSH no seu Raspberry Pi para se conectar remotamente a ele.

Configurar as informações do dispositivo para o IDT

Configure as informações do seu dispositivo para o IDT executar o teste. É preciso atualizar o modelo `device.json` localizado na pasta *`<device-tester-extract-location>`*`/configs` com as informações a seguir.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

No objeto `devices`, forneça as seguintes informações:

`id`

Um identificador exclusivo, definido pelo usuário, para o seu dispositivo.

`connectivity.ip`

O endereço IP do seu dispositivo.

`connectivity.port`

Opcional. O número da porta que deve ser usado nas conexões SSH ao seu dispositivo.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.user`

O nome de usuário usado para fazer login no seu dispositivo.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`devices.connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

> **Note**
>
> Especifique `privKeyPath` somente se `method` estiver definido como `pki`.
> Especifique `password` somente se `method` estiver definido como `password`.

Compile o pacote de teste de amostra

A pasta *`<device-tester-extract-location>`*`/samples/python` contém exemplos de arquivos de configuração, código-fonte e o SDK do cliente IDT que você pode combinar em um pacote de teste usando os scripts de compilação fornecidos. A seguinte árvore de diretórios mostra a localização desses arquivos de amostra:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
```

```
### sdks
    ### ...
    ### python
        ### idt_client
```

Para compilar o pacote de teste, execute os seguintes comandos em seu computador 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
```

Isso compila o conjunto de testes de amostra na pasta `IDTSampleSuitePython_1.0.0` dentro da pasta *`<device-tester-extract-location>`*`/tests`. Examine os arquivos na pasta `IDTSampleSuitePython_1.0.0` para entender como o pacote de teste de amostra está estruturado e para ver vários exemplos de executáveis de casos de teste e arquivos de configuração de teste.

> **Note**
>
> O pacote de teste de amostra inclui um código-fonte em Python. Não inclua informações confidenciais no código do seu pacote de teste.

Próxima etapa: use o IDT para executar o pacote de teste de amostra criada.

Usar o IDT para executar o pacote de teste de amostra

Para executar o pacote de teste de amostra, execute os seguintes comandos em seu computador host:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

O IDT executa o pacote de teste de amostra e transmite os resultados para o console. Quando a execução do teste é concluída, são vistas as seguintes informações:

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

Solução de problemas

Use as informações a seguir para ajudar a resolver os problemas ao concluir o tutorial.

O caso de teste não foi executado com êxito

- Se o teste não for executado, o IDT transmitirá os logs de erro para o console e isso pode ajudar a solucionar o problema de execução do teste. Verifique se todos os pré-requisitos deste tutorial são atendidos.

Não é possível se conectar ao dispositivo em teste

Verifique o seguinte:

- Seu arquivo `device.json` contém o endereço IP, a porta e as informações de autenticação corretos.
- Também é possível se conectar ao seu dispositivo via SSH do seu computador host.

Tutorial: desenvolva um pacote de teste simples do IDT

Um pacote de teste combina o seguinte:

- Executáveis de teste que contêm a lógica de teste
- Arquivos de configuração que descrevem o pacote de teste

Este tutorial mostra como usar o IDT para FreeRTOS para desenvolver um pacote de teste em Python contendo um caso de teste único. Embora este tutorial use SSH, é útil aprender a usar AWS IoT Device Tester com dispositivos FreeRTOS.

Neste tutorial, você concluirá as seguintes etapas:

1. Crie um diretório de pacotes de teste
2. Crie arquivos de configuração
3. Crie o executável do caso de teste
4. Execute o pacote de teste

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- Requisitos do computador host
  - Versão mais recente do AWS IoT Device Tester
  - Python 3.7 ou posterior

    Para verificar a versão do Python instalada em seu computador, execute o seguinte comando:

    ```
    python3 --version
    ```

    No Windows, se o uso deste comando retornar um erro, use `python --version`. Se o número da versão retornada for 3.7 ou superior, execute o seguinte comando em um terminal do Powershell para definir `python3` como um alias para seu comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

Se nenhuma informação sobre versão for retornada ou se o número da versão for menor do que 3.7, siga as instruções em Fazer download do Python para instalar o Python 3.7+. Para obter mais informações, consulte a Documentação do Python.

- urllib3

  Para verificar se `urllib3` está instalado corretamente, execute o seguinte comando:

  ```
  python3 -c 'import urllib3'
  ```

  Se o `urllib3` não estiver instalado, execute o comando a seguir para instalá-lo:

  ```
  python3 -m pip install urllib3
  ```

- Requisitos do dispositivo
  - Um dispositivo com sistema operacional Linux e uma conexão de rede com a mesma rede do seu computador host.

    É recomendado o uso de um Raspberry Pi com o sistema operacional Raspberry Pi. Certifique-se de configurar o SSH no seu Raspberry Pi para se conectar remotamente a ele.

Crie um diretório de pacotes de teste

O IDT separa logicamente os casos de teste em grupos de teste dentro de cada pacote de teste. Cada caso de teste deve estar dentro de um grupo de teste. Para este tutorial, crie uma pasta chamada `MyTestSuite_1.0.0` e crie a seguinte árvore de diretórios nesta pasta:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Crie arquivos de configuração

Seu pacote de teste deve conter os seguintes arquivos de configuração necessários:

Arquivos necessários

`suite.json`

Contém informações sobre o pacote de teste. Consulte Configurar suite.json.

`group.json`

Contém informações sobre um grupo de teste. É preciso criar um arquivo `group.json` para cada grupo de teste em seu pacote de teste. Consulte Configurar group.json.

`test.json`

Contém informações sobre um caso de teste. É preciso criar um arquivo `test.json` para cada caso de teste em seu pacote de teste. Consulte Configurar test.json.

1. Na pasta `MyTestSuite_1.0.0/suite`, crie um arquivo `suite.json` com a seguinte estrutura:

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. Na pasta `MyTestSuite_1.0.0/myTestGroup`, crie um arquivo `group.json` com a seguinte estrutura:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. Na pasta `MyTestSuite_1.0.0/myTestGroup/myTestCase`, crie um arquivo `test.json` com a seguinte estrutura:

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

Agora, a árvore de diretórios da pasta `MyTestSuite_1.0.0` deve ser semelhante à seguinte:

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Obtenha o SDK do cliente do IDT

Use o SDK de cliente do IDT para permitir que o IDT interaja com o dispositivo testado e relate os resultados do teste. Neste tutorial, use a versão em Python do SDK.

Na pasta *<device-tester-extract-location>*`/sdks/python/`, copie a pasta `idt_client` para sua pasta `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`.

Para verificar se o SDK foi copiado com êxito, execute o comando a seguir.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
```

```
python3 -c 'import idt_client'
```

Crie o executável do caso de teste

Os executáveis do caso de teste contêm a lógica de teste que você deseja executar. Um pacote de teste pode conter vários executáveis de casos de teste. Para este tutorial, crie somente um executável de caso de teste.

1. Crie o arquivo do pacote de teste.

   Na pasta `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`, crie um arquivo `myTestCase.py` com o seguinte conteúdo:

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

   if __name__ == "__main__":
       main()
   ```

2. Use as funções do SDK de cliente para adicionar a seguinte lógica de teste ao seu arquivo `myTestCase.py`:

   a. Execute um comando SSH no dispositivo em teste.

   ```
   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. Envie o resultado do teste para o 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()
```

Configurar as informações do dispositivo para o IDT

Configure as informações do seu dispositivo para o IDT executar o teste. É preciso atualizar o modelo `device.json` localizado na pasta *<device-tester-extract-location>*`/configs` com as informações a seguir.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

No objeto `devices`, forneça as seguintes informações:

`id`

Um identificador exclusivo, definido pelo usuário, para o seu dispositivo.

`connectivity.ip`

O endereço IP do seu dispositivo.

`connectivity.port`

Opcional. O número da porta que deve ser usado nas conexões SSH ao seu dispositivo.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.user`

O nome de usuário usado para fazer login no seu dispositivo.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`devices.connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

> **Note**
>
> Especifique `privKeyPath` somente se `method` estiver definido como `pki`.
> Especifique `password` somente se `method` estiver definido como `password`.

Execute o pacote de teste

Depois de criar o pacote de teste, verifique se ele funciona conforme o esperado. Conclua as etapas a seguir para executar o pacote de teste em seu grupo de dispositivos existente.

1. Copie sua pasta `MyTestSuite_1.0.0` em *`<device-tester-extract-location>`*`/tests`.
2. Execute os seguintes comandos:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

O IDT executa seu pacote de teste e transmite os resultados para o console. Quando a execução do teste é concluída, são vistas as seguintes informações:

```
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
```

Solução de problemas

Use as informações a seguir para ajudar a resolver os problemas ao concluir o tutorial.

O caso de teste não foi executado com êxito

Se o teste não for executado, o IDT transmitirá os logs de erro para o console e isso pode ajudar a solucionar o problema de execução do teste. Antes de verificar os logs de erro, verifique o seguinte:

- O SDK de cliente do IDT está na pasta correta, conforme descrito nesta etapa.
- Você atende a todos os pré-requisitos deste tutorial.

Não é possível se conectar ao dispositivo em teste

Verifique o seguinte:

- Seu arquivo `device.json` contém o endereço IP, a porta e as informações de autenticação corretos.
- Também é possível se conectar ao seu dispositivo via SSH do seu computador host.

Crie arquivos de configuração do pacote de teste do IDT

Esta seção descreve os formatos nos quais você cria arquivos de configuração incluídos ao escrever um pacote de teste personalizado.

Arquivos de configuração necessária

`suite.json`

Contém informações sobre o pacote de teste. Consulte Configurar suite.json.

`group.json`

Contém informações sobre um grupo de teste. É preciso criar um arquivo `group.json` para cada grupo de teste em seu pacote de teste. Consulte Configurar group.json.

`test.json`

Contém informações sobre um caso de teste. É preciso criar um arquivo `test.json` para cada caso de teste em seu pacote de teste. Consulte Configurar test.json.

Arquivos de configuração opcional

`test_orchestrator.yaml` ou **`state_machine.json`**

Define como os testes são executados quando o IDT executa o pacote de teste. Consulte Configurar test_orchestrator.yaml.

> **Note**
>
> Desde o IDT v4.5.2, você usa o arquivo `test_orchestrator.yaml` para definir o fluxo de trabalho de testes. Nas versões anteriores do IDT, você usa o arquivo

`state_machine.json`. Para obter informações sobre a máquina de estado, consulte Configurar a máquina de estado do IDT.

`userdata_schema.json`

Define o esquema do arquivo `userdata.json` que os executores de teste podem incluir na definição de configuração. O arquivo `userdata.json` é usado em qualquer informação de configuração adicional necessária para executar o teste, mas que não esteja presente no arquivo `device.json`. Consulte Configurar userdata_schema.json.

Os arquivos de configuração JSON são colocados no seu *<custom-test-suite-folder>*, conforme mostrado aqui.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configurar suite.json

O arquivo `suite.json` define as variáveis de ambiente e determina se os dados do usuário são necessários para executar o pacote de teste. Use o modelo a seguir para configurar seu arquivo *<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 os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID exclusivo, definido pelo usuário, para o pacote de teste. O valor de `id` deve corresponder ao nome da pasta do pacote de teste na qual o arquivo `suite.json` está localizado. O nome e a versão do pacote devem atender aos seguintes requisitos:

- *`<suite-name>`* não pode conter sublinhados.
- *`<suite-version>`* é indicado como *`x.x.x`*, em que `x` é um número.

O ID é mostrado nos relatórios de teste gerados pelo IDT.

`title`

Um nome definido pelo usuário para o produto ou recurso que está sendo testado por esse pacote de teste. O nome é exibido na CLI do IDT para os executores de teste.

`details`

Uma descrição breve da finalidade do pacote de teste.

`userDataRequired`

Define se os executores de teste precisam incluir informações personalizadas em um arquivo `userdata.json`. Se definir esse valor como `true`, também deverá incluir o arquivo `userdata_schema.json` na pasta do pacote de teste.

`environmentVariables`

Opcional. Uma matriz de variáveis de ambiente para configurar para esse pacote de teste.

`environmentVariables.key`

O nome da variável de ambiente.

`environmentVariables.value`

O valor da variável de ambiente.

Configurar group.json

O arquivo `group.json` define se o grupo de teste é necessário ou opcional. Use o modelo a seguir para configurar seu arquivo *<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json`:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID exclusivo, definido pelo usuário, para o grupo de teste. O valor de `id` deve corresponder ao nome da pasta do grupo de teste na qual o arquivo `group.json` está localizado e não deve ter sublinhados (_). O ID é usado nos relatórios de teste gerados pelo IDT.

`title`

Um nome descritivo para o grupo de teste. O nome é exibido na CLI do IDT para os executores de teste.

`details`

Uma descrição breve da finalidade do grupo de teste.

`optional`

Opcional. Defina como `true` para exibir este grupo de teste como um grupo opcional depois que o IDT terminar de executar os testes necessários. O valor padrão é `false`.

Configurar test.json

O arquivo `test.json` determina os executáveis do caso de teste e as variáveis de ambiente que são usadas por um caso de teste. Para obter mais informações sobre como criar executáveis de casos de teste, consulte Crie um executável de caso de teste do IDT.

Use o modelo a seguir para configurar seu arquivo *<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 os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID exclusivo, definido pelo usuário, para o caso de teste. O valor de `id` deve corresponder ao nome da pasta do caso de teste na qual o arquivo `test.json` está localizado e não deve ter sublinhados (_). O ID é usado nos relatórios de teste gerados pelo IDT.

`title`

Um nome descritivo para o caso de teste. O nome é exibido na CLI do IDT para os executores de teste.

`details`

Uma descrição breve da finalidade do caso de teste.

`requireDUT`

Opcional. Defina como `true` se um dispositivo for necessário para executar este teste. Caso contrário, defina como `false`. O valor padrão é `true`. Os executores de teste configurarão os dispositivos que usarão para executar o teste nos arquivos `device.json`.

`requiredResources`

Opcional. Uma matriz que fornece informações sobre os dispositivos de recursos necessários para executar esse teste.

`requiredResources.name`

O nome exclusivo a ser dado ao dispositivo de recurso quando este teste está sendo executado.

`requiredResources.features`

Uma matriz de recursos de dispositivos de recursos definidos pelo usuário.

`requiredResources.features.name`

O nome do recurso. O recurso do dispositivo para o qual você deseja usar este dispositivo. Este nome é comparado ao nome do recurso fornecido pelo executor de teste no arquivo `resource.json`.

`requiredResources.features.version`

Opcional. A versão do recurso. Este valor é comparado à versão do recurso fornecida pelo executor de teste no arquivo `resource.json`. Se uma versão não for fornecida, o recurso não será verificado. Se um número de versão não for obrigatório para o recurso, deixe este campo em branco.

`requiredResources.features.jobSlots`

Opcional. O número de testes simultâneos que podem ser compatíveis com este recurso. O valor padrão é 1. Se você quiser que o IDT use dispositivos distintos para recursos individuais, recomendamos que você defina esse valor como 1.

`execution.timeout`

A quantidade de tempo (em milissegundos) que o IDT aguardará a conclusão da execução do teste. Para obter mais informações sobre este valor, consulte Crie um executável de caso de teste do IDT.

`execution.os`

Os executáveis do caso de teste a serem executados com base no sistema operacional do computador host que executa o IDT. Os valores compatíveis são `linux`, `mac` e `win`.

`execution.os.cmd`

O caminho para o executável do caso de teste que deseja executar para o sistema operacional especificado. Este local deve estar no caminho do sistema.

`execution.os.args`

Opcional. Os argumentos a serem fornecidos para executar o executável do caso de teste.

`environmentVariables`

Opcional. Uma matriz de variáveis de ambiente definidas para este caso de teste.

`environmentVariables.key`

O nome da variável de ambiente.

`environmentVariables.value`

O valor da variável de ambiente.

> **Note**
>
> Se especificar a mesma variável de ambiente no arquivo `test.json` e no arquivo `suite.json`, o valor no arquivo `test.json` terá precedência.

Configurar test_orchestrator.yaml

Um orquestrador de teste é uma estrutura que controla o fluxo de execução do pacote de teste. Ele determina o estado inicial de um pacote de teste, gerencia as transições de estado com base nas regras definidas pelo usuário e continua a transição por esses estados até atingir o estado final.

Se seu pacote de teste não incluir um orquestrador de testes definido pelo usuário, o IDT gerará um orquestrador de testes para você.

O orquestrador de teste padrão executa as seguintes funções:

- Fornece aos executores de teste a capacidade de selecionar e executar grupos de testes específicos, em vez de todo o pacote de teste.
- Se grupos de teste específicos não forem selecionados, executará cada grupo de teste no pacote de teste em uma ordem aleatória.
- Gera relatórios e imprime um resumo do console que mostra os resultados do teste para cada grupo de teste e caso de teste.

Para obter mais informações sobre como o orquestrador de testes do IDT funciona, consulte Configure o orquestrador de testes do IDT.

Configurar userdata_schema.json

O arquivo `userdata_schema.json` determina o esquema no qual os executores de teste fornecem dados do usuário. Os dados do usuário são necessários se seu pacote de teste exigir informações que não estejam presentes no arquivo `device.json`. Por exemplo, seus testes podem precisar de credenciais de rede Wi-Fi, portas abertas específicas ou certificados que um usuário deve fornecer. Estas informações podem ser fornecidas ao IDT como um parâmetro de entrada chamado `userdata`, cujo valor é um arquivo `userdata.json`, que os usuários criam em suas pastas *`<device-tester-extract-location>`*`/config`. O formato do arquivo `userdata.json` é baseado no arquivo `userdata_schema.json` incluído no pacote de teste.

Para indicar que os executores de teste devem fornecer um arquivo `userdata.json`:

1. No arquivo `suite.json`, defina `userDataRequired` como `true`.
2. No seu *`<custom-test-suite-folder>`*, crie um arquivo `userdata_schema.json`.
3. Edite o arquivo `userdata_schema.json` para criar um esquema JSON válido do IETF Draft v4.

Quando o IDT executa seu pacote de teste, ele lê automaticamente o esquema e o usa para validar o arquivo `userdata.json` fornecido pelo executor do teste. Se válido, o conteúdo do arquivo `userdata.json` estará disponível no contexto do IDT e no contexto do orquestrador de testes.

Configure o orquestrador de testes do IDT

Desde o IDT v4.5.2, o IDT inclui um novo componente orquestrador de testes. O orquestrador de testes é um componente do IDT que controla o fluxo de execução do pacote de teste e gera o relatório de teste depois que o IDT termina de executar todos os testes. O orquestrador de testes determina a seleção do teste e a ordem na qual os testes são executados com base nas regras definidas pelo usuário.

Se seu pacote de teste não incluir um orquestrador de testes definido pelo usuário, o IDT gerará um orquestrador de testes para você.

O orquestrador de teste padrão executa as seguintes funções:

- Fornece aos executores de teste a capacidade de selecionar e executar grupos de testes específicos, em vez de todo o pacote de teste.
- Se grupos de teste específicos não forem selecionados, executará cada grupo de teste no pacote de teste em uma ordem aleatória.
- Gera relatórios e imprime um resumo do console que mostra os resultados do teste para cada grupo de teste e caso de teste.

O orquestrador de teste substitui a máquina de estado do IDT. É recomendado utilizar a versão mais recente do orquestrador de testes para desenvolver seus pacotes de teste em vez da máquina de estado do IDT. O orquestrador de testes fornece os seguintes recursos aprimorados:

- Usa um formato declarativo em comparação com o formato imperativo que a máquina de estado do IDT usa. Isto permite que especificar quais testes deseja executar e quando deseja executá-los.
- Gerencia o tratamento de grupos específicos, a geração de relatórios, o tratamento de erros e o rastreamento de resultados para que você não precise gerenciar manualmente essas ações.

- Usa o formato YAML, que é compatível com comentários por padrão.
- Requer 80% menos espaço em disco do que o orquestrador de teste para definir o mesmo fluxo de trabalho.
- Adiciona validação de pré-teste para verificar se a definição do fluxo de trabalho não contém IDs de teste incorretos ou dependências circulares.

Formato do orquestrador de testes

É possível usar o modelo a seguir para configurar seu próprio arquivo *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 os campos que contêm valores são necessários, conforme descrito aqui:

`Aliases`

Opcional. Strings definidas pelo usuário que são mapeadas para expressões de contexto. Os aliases permitem que você gere nomes amigáveis para identificar expressões de contexto na configuração do orquestrador de teste. Isto é muito útil se estiver criando expressões de contexto complexas ou expressões usadas em vários lugares.

É possível usar expressões de contexto para armazenar consultas de contexto que permitem acessar dados de outras configurações do IDT. Para ter mais informações, consulte Acesse dados no contexto.

Example

Exemplo

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

Opcional. Uma lista de condições e os casos de teste correspondentes que são executados quando cada condição é atendida. Cada condição pode ter vários casos de teste. No entanto, só é possível atribuir um determinado caso de teste a uma condição.

Por padrão, o IDT executa qualquer caso de teste que não esteja atribuído a uma condição nessa lista. Se não especificar essa seção, o IDT executará todos os grupos de teste no pacote de teste.

Cada item da lista `ConditionalTests` inclui os seguintes parâmetros:

`Condition`

Uma expressão de contexto que deve ser avaliada para um valor booleano. Se o valor avaliado for verdadeiro, o IDT executará os casos de teste especificados no parâmetro `Tests`.

`Tests`

A lista dos descritores de teste.

Cada descritor de testes usa o ID do grupo de teste e um ou mais IDs do caso de teste para identificar os testes individuais a serem executados a partir de um grupo de teste específico. O descritor de teste usa o seguinte formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example

Exemplo

O exemplo a seguir usa expressões de contexto genéricas que podem ser definidas 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
```

Com base nas condições definidas, o IDT seleciona os grupos de teste da seguinte forma:

- Se `Condition1` for verdade, o IDT executa os testes nos grupos de teste A, B e C.
- Se `Condition2` for verdade, o IDT executa os testes nos grupos de teste C e D.

`Order`

Opcional. A ordem em que os testes serão executados. Especifique a ordem do teste no nível do grupo de teste. Se não especificar essa seção, o IDT executará todos os grupos de teste aplicáveis em uma ordem aleatória. O valor de `Order` é uma lista de listas de descritores de grupos. Qualquer grupo de teste que não esteja listado em `Order` pode ser executado em paralelo com qualquer outro grupo de teste listado.

Cada lista de descritores de grupo contém um ou mais descritores de grupo e identifica a ordem na qual executar os grupos especificados em cada descritor. É possível usar os seguintes formatos para definir descritores de grupo individuais:

- *`group-id`*: o ID do grupo de um grupo de teste existente.
- [*`group-id`*, *`group-id`*]: lista de grupos de teste que podem ser executados em qualquer ordem em relação um ao outro.
- `"*"`: curinga. Isto é equivalente à lista de todos os grupos de teste que ainda não estão especificados na lista atual de descritores de grupo.

O valor de `Order` também devem atender aos seguintes requisitos:

- Os IDs de grupo de teste que você especifica em um descritor de grupo devem existir em seu pacote de teste.
- Cada lista de descritores de grupo deve incluir pelo menos um grupo de teste.
- Cada lista de descritores de grupo deve conter IDs de grupo exclusivos. Não é possível repetir um ID de grupo de teste em descritores de grupos individuais.
- Uma lista de descritores de grupo pode ter no máximo um descritor de grupo curinga. O descritor do grupo curinga deve ser o primeiro ou o último item na lista.

Example

Exemplo

Para um pacote de teste que contém os grupos de teste A, B, C, D e E, a lista de exemplos a seguir mostra maneiras diferentes de especificar que o IDT deve primeiro executar o grupo de teste A, depois executar o grupo de teste B e, em seguida, executar os grupos de teste C, D e E em qualquer ordem.

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```
  ```

`Features`

Opcional. A lista de recursos do produto que você deseja que o IDT adicione ao arquivo `awsiotdevicetester_report.xml`. Se você não especificar essa seção, o IDT não adicionará nenhum recurso do produto ao relatório.

Um recurso do produto é uma informação definida pelo usuário sobre critérios específicos que um dispositivo pode atender. Por exemplo, o recurso do produto MQTT pode indicar que o dispositivo publica mensagens MQTT corretamente. Em `awsiotdevicetester_report.xml`, os recursos do produto são definidos como `supported`, `not-supported` ou um valor personalizado e definido pelo usuário, com base na aprovação dos testes especificados.

Cada item da lista `Features` consiste dos seguintes parâmetros:

`Name`

O nome do recurso.

`Value`

Opcional. O valor personalizado que deseja usar no relatório em vez de `supported`. Se esse valor não for especificado, com base nos conjuntos do IDT, o valor do recurso será definido como `supported` ou `not-supported`. Se você testar o mesmo recurso com condições diferentes, poderá usar um valor personalizado para cada instância desse recurso na lista `Features`, e o IDT concatena os valores do recurso para as condições compatíveis. Para obter mais informações, consulte

`Condition`

Uma expressão de contexto que deve ser avaliada para um valor booleano. Se o valor avaliado for verdadeiro, o IDT adiciona o recurso ao relatório de teste após concluir a execução do pacote de teste. Se o valor avaliado for falso, o teste não será incluído no relatório.

`Tests`

Opcional. A lista dos descritores de teste. Todos os testes que são especificados nesta lista devem ser aprovados para que o recurso seja compatível.

Cada descritor de testes nesta lista usa o ID do grupo de teste e um ou mais IDs do caso de teste para identificar os testes individuais a serem executados em um grupo de teste específico. O descritor de teste usa o seguinte formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Você deve especificar um `Tests` ou `OneOfTests` para cada recurso na lista `Features`.

`OneOfTests`

Opcional. A lista dos descritores de teste. Pelo menos um dos testes especificados nesta lista deve ser aprovado para que o recurso seja compatível.

Cada descritor de testes nesta lista usa o ID do grupo de teste e um ou mais IDs do caso de teste para identificar os testes individuais a serem executados em um grupo de teste específico. O descritor de teste usa o seguinte formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Você deve especificar um `Tests` ou `OneOfTests` para cada recurso na lista `Features`.

`IsRequired`

O valor booleano que define se o recurso é exigido no relatório de teste. O valor padrão é `false`.

Contexto do orquestrador de teste

O contexto do orquestrador de estado é um documento JSON somente para leitura que contém dados que estão disponíveis para o orquestrador de estado durante a execução. O contexto do orquestrador de estado é acessível somente do orquestrador de estado e contém informações que determinam o fluxo de teste. Por exemplo, você pode usar as informações configuradas pelos executores de teste no arquivo `userdata.json` para determinar se é necessário executar um teste específico.

O contexto do orquestrador de teste usa o seguinte formato:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
```

```
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`pool`

Informações sobre o grupo de dispositivos selecionado para a execução do teste. Para um grupo de dispositivos selecionado, essas informações são recuperadas do elemento correspondente da matriz do grupo de dispositivos de nível superior definido no arquivo `device.json`.

`userData`

As informações no arquivo `userdata.json`.

`config`

As informações no arquivo `config.json`.

É possível consultar o contexto usando a notação JSONPath. A sintaxe para consultas JSONPath nas definições de estado é `{{query}}`. Ao acessar dados do contexto do orquestrador de testes, verifique que cada valor seja avaliado como uma string, um número ou um booleano.

Para obter mais informações sobre o uso da notação JSONPath para acessar dados do contexto, consulte Use o contexto do IDT.

Configurar a máquina de estado do IDT

> **Important**
>
> Desde o IDT v4.5.2, esta máquina de estado está obsoleta. É muito recomendado o uso do novo orquestrador de testes. Para ter mais informações, consulte Configure o orquestrador de testes do IDT.

Uma máquina de estado é uma estrutura que controla o fluxo de execução do pacote de teste. Ele determina o estado inicial de um pacote de teste, gerencia as transições de estado com base nas regras definidas pelo usuário e continua a transição por esses estados até atingir o estado final.

Se seu pacote de teste não incluir uma máquina de estado definida pelo usuário, o IDT gerará uma máquina de estado para você. A máquina de estado padrão executa as seguintes funções:

- Fornece aos executores de teste a capacidade de selecionar e executar grupos de testes específicos, em vez de todo o pacote de teste.
- Se grupos de teste específicos não forem selecionados, executará cada grupo de teste no pacote de teste em uma ordem aleatória.
- Gera relatórios e imprime um resumo do console que mostra os resultados do teste para cada grupo de teste e caso de teste.

A máquina de estado de um pacote de teste da IDT deve atender aos seguintes critérios:

- Cada estado corresponde a uma ação a ser executada pelo IDT, como executar um grupo de teste ou produzir um arquivo de relatório.
- A transição para um estado executa a ação associada ao estado.
- Cada estado define a regra de transição para o próximo estado.
- O estado final deve ser `Succeed` ou `Fail`.

Formato da máquina de estado

É possível usar o modelo a seguir para configurar seu próprio arquivo *<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 os campos que contêm valores são necessários, conforme descrito aqui:

`Comment`

Uma descrição da máquina de estado.

`StartAt`

O nome do estado em que o IDT começa a executar o pacote de teste. O valor de `StartAt` deve ser definido como um dos estados listados no objeto `States`.

`States`

Um objeto que mapeia nomes de estados definidos pelo usuário para estados IDT válidos. Cada objeto de estados *`.state-name`* contém a definição de um estado válido mapeado para o *`nome do estado`*.

O objeto `States` deve incluir os estados `Succeed` e `Fail`. Para obter informações sobre estados válidos, consulte Estados válidos e definições de estado.

Estados válidos e definições de estado

Esta seção descreve as definições de estado de todos os estados válidos que podem ser usados na máquina de estados IDT. Alguns dos estados a seguir são compatíveis com configurações no nível do caso de teste. No entanto, é recomendado configurar as regras de transição de estado no nível do grupo de teste em vez do nível do caso de teste, a menos que seja absolutamente necessário.

Definições do estado

- RunTask
- Escolha
- Paralelo
- AddProductFeatures
- Relatório
- LogMessage
- SelectGroup
- Falha

- Êxito

RunTask

O estado `RunTask` executa casos de teste de um grupo de teste definido no pacote de teste.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`TestGroup`

Opcional. O ID do grupo de teste a ser executado. Se este valor não for especificado, o IDT executará o grupo de teste selecionado pelo executor de testes.

`TestCases`

Opcional. Uma matriz de IDs de casos de teste do grupo especificado em `TestGroup`. Com base nos valores de `TestGroup` e `TestCases`, o IDT determina o comportamento da execução do teste da seguinte forma:

- Quando tanto o `TestGroup` como os `TestCases` são especificados, o IDT executa os casos de teste especificados do grupo de teste.
- Quando `TestCases` são especificados, mas `TestGroup` não são especificados, o IDT executa os casos de teste especificados.
- Quando `TestGroup` é especificado, mas `TestCases` não são especificados, o IDT executa os casos no grupo de teste especificado.
- Quando nem `TestGroup` nem `TestCases` são especificados, o IDT executa todos os casos de teste do grupo de teste que o executor de teste seleciona na CLI do IDT. Para habilitar a

seleção de grupos para executores de teste, é preciso incluir ambos os estados `RunTask` e `Choice` em seu arquivo `statemachine.json`. Para ver um exemplo de como isso funciona, consulte Exemplo de máquina de estado: executar grupos de teste selecionados pelo usuário.

Para obter mais informações sobre como habilitar os comandos da CLI no IDT, consulte the section called “Habilitar comandos da CLI no IDT”.

`ResultVar`

O nome da variável de contexto a ser definida com os resultados da execução do teste. Não especifique este valor se você não especificou um valor para `TestGroup`. O IDT define o valor da variável que você define em `ResultVar` para `true` ou `false` com base no seguinte:

- Se o nome da variável estiver no formato *`text`*`_`*`text`*`_passed`, o valor será definido como se todos os testes do primeiro grupo de teste foram aprovados ou ignorados.
- Em todos os outros casos, o valor é definido como se todos os testes em todos os grupos de teste tivessem sido aprovados ou ignorados.

Normalmente, você usará o estado `RunTask` para especificar um ID de grupo de teste sem especificar IDs de caso de teste individuais, para que o IDT execute todos os casos de teste no grupo de teste especificado. Todos os casos de teste executados por esse estado são executados em paralelo, em uma ordem aleatória. No entanto, se todos os casos de teste exigirem a execução de um dispositivo e apenas um único dispositivo estiver disponível, os casos de teste serão executados em sequência.

Como tratar erros

Se algum dos grupos de teste ou IDs de casos de teste especificados não for válido, esse estado emitirá o erro de execução `RunTaskError`. Se o estado encontrar um erro de execução, ele também definirá a variável `hasExecutionError` no contexto da máquina de estado como `true`.

Escolha

O estado `Choice` permite definir dinamicamente o próximo estado para o qual fazer a transição com base em condições definidas pelo usuário.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
```

```
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Default`

O estado padrão para o qual fazer a transição se nenhuma das expressões definidas em `Choices` puder ser avaliada como `true`.

`FallthroughOnError`

Opcional. Especifica o comportamento quando o estado se depara com um erro na avaliação de expressões. Defina como `true` se deseja ignorar uma expressão, caso a avaliação resulte em um erro. Se não houver correspondência de expressão, a máquina de estado fará a transição para o estado `Default`. Se o valor `FallthroughOnError` não for especificado, o valor padrão será `false`.

`Choices`

Uma matriz de expressões e estados para determinar para qual estado fazer a transição depois de executar as ações no estado atual.

`Choices.Expression`

Uma string de expressão que deve ser avaliada para um valor booleano. Se a expressão for avaliada como `true`, a máquina de estado fará a transição para o estado definido em `Choices.Next`. As strings de expressão recuperam valores do contexto da máquina de estado e executam operações neles para chegar a um valor booleano. Para obter informações sobre como acessar o contexto da máquina de estado, consulte Contexto da máquina de estado.

`Choices.Next`

O nome do estado para o qual fazer a transição se a expressão definida em `Choices.Expression` for avaliada como `true`.

Como tratar erros

O estado `Choice` pode exigir o tratamento de erros nos seguintes casos:

- Algumas variáveis nas expressões de escolha não existem no contexto da máquina de estado.
- O resultado de uma expressão não é um valor booleano.
- O resultado de uma pesquisa JSON não é uma string, número ou booleano.

Não é possível usar um bloco `Catch` para tratar erros nesse estado. Se quiser parar de executar a máquina de estado quando ela encontrar um erro, defina `FallthroughOnError` como `false`. No entanto, é recomendado configurar `FallthroughOnError` e `true`, e dependendo do seu caso de uso, executar uma das seguintes etapas:

- Se uma variável que você está acessando não existir em alguns casos, use o valor de `Default` e blocos `Choices` adicionais para especificar o próximo estado.
- Se uma variável que estiver acessando sempre existir, defina o estado `Default` como `Fail`.

Paralelo

O estado `Parallel` permite que você defina e execute novas máquinas de estado em paralelo.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`Branches`

Uma matriz de definições de máquina de estado a serem executadas. Cada definição de máquina de estado deve conter os próprios estados `StartAt`, `Succeed` e `Fail`. As definições de máquina de estado nesta matriz não podem fazer referência a estados de fora da definição deles.

> **Note**
>
> Como cada máquina de estado de ramificação compartilha o mesmo contexto de máquina de estado, definir variáveis em uma ramificação e depois lê-las de outra ramificação pode resultar em um comportamento inesperado.

O estado `Parallel` passa para o próximo estado somente depois de executar todas as máquinas de estado da ramificação. Todo estado que necessita um dispositivo aguardará para ser executado até que o dispositivo esteja disponível. Se vários dispositivos estiverem disponíveis, este estado executará casos de teste a partir de vários grupos em paralelo. Se não houver dispositivos suficientes disponíveis, os casos de teste serão executados em sequência. Como os casos de teste são executados em uma ordem aleatória quando executados em paralelo, podem ser usados dispositivos diferentes para executar testes a partir do mesmo grupo de teste.

Como tratar erros

Certifique-se que tanto a máquina de estado da ramificação quanto a máquina de estado pai fazem a transição para o estado `Fail` para tratar erros de execução.

Como as máquinas de estado de ramificação não transmitem erros de execução para a máquina de estado principal, você não pode usar um bloco `Catch` para lidar com erros de execução em máquinas de estado de ramificação. Em vez disso, use o valor `hasExecutionErrors` no contexto da máquina de estado compartilhada. Para obter um exemplo de como isso funciona, consulte Exemplo de máquina de estado: execute dois grupos de testes em paralelo.

AddProductFeatures

O estado `AddProductFeatures` permite adicionar recursos do produto ao arquivo `awsiotdevicetester_report.xml` gerado pelo IDT.

Um recurso do produto é uma informação definida pelo usuário sobre critérios específicos que um dispositivo pode atender. Por exemplo, o recurso do produto `MQTT` pode indicar que o dispositivo publica mensagens MQTT corretamente. No relatório, os recursos do produto são definidas como `supported`, `not-supported` ou um valor personalizado, com base na aprovação dos testes especificados.

> **Note**
>
> O estado `AddProductFeatures` não gera relatórios por conta própria. Esse estado deve passar para o estado `Report` para gerar relatórios.

```
{
    "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 os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`Features`

Uma matriz de recursos do produto para mostrar no arquivo `awsiotdevicetester_report.xml`.

`Feature`

O nome do recurso

`FeatureValue`

Opcional. O valor personalizado a ser usado no relatório em vez de `supported`. Se esse valor não for especificado, com base nos resultados do teste, o valor do recurso será definido como `supported` ou `not-supported`.

Se você usar um valor personalizado para `FeatureValue`, poderá testar o mesmo recurso com condições diferentes e o IDT concatenará os valores do recurso para as condições compatíveis. Por exemplo, o trecho a seguir mostra o recurso `MyFeature` com dois valores de recurso separados:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Se os dois grupos de teste forem aprovados, o valor do recurso será definido como `first-feature-supported, second-feature-supported`.

`Groups`

Opcional. Uma matriz de IDs de grupos de teste. Todos os testes dentro de cada grupo de teste especificado devem ser aprovados para que o recurso seja compatível.

`OneOfGroups`

Opcional. Uma matriz de IDs de grupos de teste. Todos os testes dentro de pelo menos um dos grupos de teste especificados devem ser aprovados para que o recurso seja compatível.

`TestCases`

Opcional. Uma matriz de IDs de casos de teste. Se especificar este valor, o seguinte se aplica:

- Todos os casos de teste especificados devem ser aprovados para que o recurso seja compatível.

- `Groups` deve conter somente um ID de grupo de teste.
- `OneOfGroups` não deve ser especificado.

`IsRequired`

Opcional. Defina como `false` para marcar este recurso como um recurso opcional no relatório. O valor padrão é `true`.

`ExecutionMethods`

Opcional. Uma matriz de métodos de execução que correspondem ao valor `protocol` especificado no arquivo `device.json`. Se esse valor for especificado, os executores de teste deverão especificar um valor `protocol` que corresponda a um dos valores dessa matriz para incluir o recurso no relatório. Se esse valor não for especificado, o recurso sempre será incluído no relatório.

Para usar o estado `AddProductFeatures`, você deve definir o valor de `ResultVar` no estado `RunTask` como um dos seguintes valores:

- Se especificou IDs de casos de teste individuais, defina `ResultVar` como *`group-id_test-id`*`_passed`.
- Se não especificou IDs de casos de teste individuais, defina `ResultVar` como *`group-id`*`_passed`.

O estado `AddProductFeatures` verifica os resultados dos testes da seguinte maneira:

- Se não especificou IDs de casos de teste, o resultado de cada grupo de teste será determinado a partir do valor da variável *`group-id`*`_passed` no contexto da máquina de estado.
- Se especificou IDs de casos de teste, o resultado de cada um dos testes será determinado a partir do valor da variável *`group-id_test-id`*`_passed` no contexto da máquina de estado.

Como tratar erros

Se um ID de grupo fornecido neste estado não for um ID de grupo válido, este estado resultará no erro de execução `AddProductFeaturesError`. Se o estado encontrar um erro de execução, ele também definirá a variável `hasExecutionErrors` no contexto da máquina de estado como `true`.

Relatório

O estado `Report` gera os arquivos *`suite-name`*`_Report.xml` e `awsiotdevicetester_report.xml`. Este estado também transmite o relatório para o console.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

É sempre necessário fazer a transição para o estado `Report` perto do final do fluxo de execução do teste para que os executores de teste possam visualizar os resultados do teste. Normalmente, o próximo estado após este estado é `Succeed`.

Como tratar erros

Se este estado se deparar com problemas ao gerar relatórios, ele emitirá o erro de execução `ReportError`.

LogMessage

O estado `LogMessage` gera o arquivo `test_manager.log` e transmite a mensagem de log para o console.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`Level`

O nível de erro no qual criar a mensagem de log. Se especificar um nível que não seja válido, este estado irá gerar uma mensagem de erro e a descartará.

`Message`

A mensagem a ser registrada.

SelectGroup

O estado `SelectGroup` atualiza o contexto da máquina de estado para indicar quais grupos estão selecionados. Os valores definidos por esse estado são usados por qualquer estado `Choice` seguinte.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`TestGroups`

Uma matriz de grupos de teste que serão marcados como selecionados. Para cada ID de grupo de teste nesta matriz, a variável *`group-id`*`_selected` é definida como `true` no contexto. Certifique-se de fornecer IDs de grupos de teste válidos porque o IDT não valida se os grupos especificados existem.

Falha

O estado `Fail` indica que a máquina de estado não foi executada corretamente. Este é um estado final para a máquina de estados e cada definição de máquina de estado deve incluir este estado.

```
{
```

```
    "Type": "Fail"
}
```

Êxito

O estado `Succeed` indica que a máquina de estado foi executada corretamente. Este é um estado final para a máquina de estados e cada definição de máquina de estado deve incluir este estado.

```
{
    "Type": "Succeed"
}
```

Contexto da máquina de estado

O contexto da máquina de estado é um documento JSON somente para leitura que contém dados que estão disponíveis para a máquina de estado durante a execução. O contexto da máquina de estado é acessível somente da máquina de estado e contém informações que determinam o fluxo de teste. Por exemplo, você pode usar as informações configuradas pelos executores de teste no arquivo `userdata.json` para determinar se é necessário executar um teste específico.

O contexto da máquina de estado usa o seguinte 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`

Informações sobre o grupo de dispositivos selecionado para a execução do teste. Para um grupo de dispositivos selecionado, essas informações são recuperadas do elemento correspondente da matriz do grupo de dispositivos de nível superior definido no arquivo `device.json`.

`userData`

As informações no arquivo `userdata.json`.

`config`

As informações fixadas no arquivo `config.json`.

`suiteFailed`

O valor é definido como `false` quando a máquina de estado for iniciada. Se um grupo de teste falhar em um estado `RunTask`, este valor será definido como `true` para a duração restante da execução da máquina de estado.

`specificTestGroups`

Se o executor de teste selecionar grupos de teste específicos para execução em vez de todo o pacote de teste, esta chave será criada e conterá a lista de IDs de grupos de teste específicos.

`specificTestCases`

Se o executor de teste selecionar casos de teste específicos para execução em vez de todo o pacote de teste, esta chave será criada e conterá a lista de IDs de casos de teste específicos.

`hasExecutionErrors`

Não saia quando a máquina de estado é iniciada. Se algum estado encontrar um erro de execução, esta variável será criada e definida como `true` para a duração restante da execução da máquina de estado.

É possível consultar o contexto usando a notação JSONPath. A sintaxe para consultas JSONPath nas definições de estado é `{{$.query}}`. Você pode usar consultas JSONPath como strings de espaço reservado em alguns estados. O IDT substitui as strings de espaço reservado pelo valor da consulta JSONPath avaliada no contexto. É possível usar espaços reservados para os seguintes valores:

- O valor `TestCases` nos estados `RunTask`.
- O valor de `Expression` no estado `Choice`.

Ao acessar dados do contexto da máquina de estado, verifique se as seguintes condições são atendidas:

- Seus caminhos JSON devem começar com `$.`
- Cada valor deve ser avaliado como uma string, um número ou um booleano.

Para obter mais informações sobre o uso da notação JSONPath para acessar dados do contexto, consulte Use o contexto do IDT.

Erros de execução

Os erros de execução são erros na definição da máquina de estado que a máquina de estado encontra ao executar um estado. O IDT registra informações sobre cada erro no arquivo `test_manager.log` e transmite a mensagem de log para o console.

É possível usar os seguintes métodos para lidar com erros de execução:

- Adicione um bloco `Catch` na definição do estado.
- Verifique o valor do valor `hasExecutionErrors` no contexto da máquina de estado.

Catch

Para usar `Catch`, adicione o seguinte à sua definição de estado:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Catch.ErrorEquals`

Uma matriz dos tipos de erro a serem capturados. Se um erro de execução corresponder a um dos valores especificados, a máquina de estado fará a transição para o estado especificado em

`Catch.Next`. Consulte cada definição de estado para obter informações sobre o tipo de erro que ela produz.

`Catch.Next`

O próximo estado para o qual fazer a transição se o estado atual encontrar um erro de execução que corresponder a um dos valores especificados em `Catch.ErrorEquals`.

Os blocos de captura são manuseados de maneira sequencial até que um deles corresponda. Se os erros não corresponderem aos listados nos blocos Catch, as máquinas de estado continuarão a ser executadas. Como os erros de execução são resultado de definições de estado incorretas, recomendamos que você faça a transição para o estado Falha quando um estado encontrar um erro de execução.

hasExecutionError

Quando alguns estados encontram erros de execução, além de emitirem o erro, eles também definem o valor `hasExecutionError` como `true` no contexto da máquina de estado. É possível usar este valor para detectar quando ocorre um erro e, em seguida, usar um estado `Choice` para fazer a transição da máquina de estado para o estado `Fail`.

Este método tem as características a seguir.

- A máquina de estado não inicia com nenhum valor atribuído para `hasExecutionError` e este valor não está disponível até que um determinado estado a defina. Isto significa que é preciso definir explicitamente o `FallthroughOnError` como `false` para os estados `Choice` que acessam este valor para evitar que a máquina de estado pare se nenhum erro de execução ocorrer.
- Depois de definido como `true`, `hasExecutionError` nunca é definido como falso ou removido do contexto. Isto significa que esse valor é útil somente na primeira vez em que é definido como `true` e, para todos os estados subsequentes, não fornece um valor significativo.
- O valor `hasExecutionError` é compartilhado com todas as máquinas de estado da filial no estado`Parallel`, o que pode resultar em resultados inesperados, dependendo da ordem em que é acessado.

Por conta dessas características, não é recomendado usar este método, e sim, usar um bloco Catch.

Exemplo de máquinas de estado

Esta seção fornece alguns exemplos de configurações de máquina de estado.

Exemplos

- Exemplo de máquina de estado: execute um único grupo de teste
- Exemplo de máquina de estado: execute grupos de teste selecionados pelo usuário
- Exemplo de máquina de estado: execute um único grupo de teste com recursos do produto
- Exemplo de máquina de estado: execute dois grupos de testes em paralelo

Exemplo de máquina de estado: execute um único grupo de teste

Esta máquina de estado:

- Executa o grupo de teste com o ID `GroupA`, que deve estar presente no pacote em um arquivo `group.json`.
- Verifica se há erros de execução e transições de `Fail` para ver se algum foi encontrado.
- Gera um relatório e faz a transição para `Succeed` se não houver erros e, caso contrário, `Fail`.

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

Exemplo de máquina de estado: execute grupos de teste selecionados pelo usuário

Esta máquina de estado:

- Verifica se o executor do teste selecionou grupos de teste específicos. A máquina de estado não verifica casos de teste específicos porque os executores de teste não podem selecionar casos de teste sem também selecionar um grupo de teste.
- Se os grupos de teste forem selecionados:
  - Executa os casos de teste nos grupos de teste selecionados. Para fazer isso, a máquina de estado não especifica explicitamente nenhum grupo de teste ou caso de teste no estado `RunTask`.
  - Gera um relatório após executar todos os testes e sai.
- Se os grupos de teste não forem selecionados:
  - Executa testes no grupo de teste `GroupA`.
  - Gera relatórios e sai.

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

Exemplo de máquina de estado: execute um único grupo de teste com recursos do produto

Esta máquina de estado:

- Executa o grupo de teste `GroupA`.
- Verifica se há erros de execução e transições de `Fail` para ver se algum foi encontrado.
- Adiciona o recurso `FeatureThatDependsOnGroupA` ao arquivo `awsiotdevicetester_report.xml`:
  - Se `GroupA` for aprovado, o recurso será definido como `supported`.
  - O recurso não está marcado como opcional no relatório.
- Gera um relatório e faz a transição para `Succeed` se não houver erros e, caso contrário, `Fail`

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

Exemplo de máquina de estado: execute dois grupos de testes em paralelo

Esta máquina de estado:

- Executa os grupos de teste `GroupA` e `GroupB` em paralelo. As variáveis `ResultVar` armazenadas no contexto pelos estados `RunTask` nas máquinas de estado da ramificação estão disponíveis para o estado `AddProductFeatures`.
- Verifica se há erros de execução e transições de `Fail` para ver se algum foi encontrado. Esta máquina de estado não usa um bloco `Catch` porque este método não detecta erros de execução em máquinas de estado de ramificação.
- Adiciona recursos ao arquivo `awsiotdevicetester_report.xml` com base nos grupos que passam
  - Se `GroupA` for aprovado, o recurso será definido como `supported`.
  - O recurso não está marcado como opcional no relatório.
- Gera um relatório e faz a transição para `Succeed` se não houver erros e, caso contrário, `Fail`

Se dois dispositivos estiverem configurados no grupo de dispositivos, tanto o `GroupA` como o `GroupB` poderão ser executados ao mesmo tempo. No entanto, se um `GroupA` ou `GroupB` tiver vários testes, os dois dispositivos poderão ser alocados para esses testes. Se somente um dispositivo estiver configurado, os grupos de teste serão executados em sequência.

```
{
    "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"
```

```
        }
    }
}
```

Crie um executável de caso de teste do IDT

É possível criar e colocar o executável do caso de teste em uma pasta do pacote de teste das seguintes maneiras:

- Para pacotes de testes que usam argumentos ou variáveis de ambiente dos arquivos `test.json` para determinar quais testes executar, você pode criar um único caso de teste executável para todo o pacote de teste ou um executável de teste para cada grupo de teste no pacote de teste.
- Para um pacote de teste em que você deseja executar testes específicos com base em comandos especificados, você cria um caso de teste executável para cada caso de teste no pacote de teste.

Como redator de teste, é possível determinar qual abordagem é apropriada para seu caso de uso e estruturar o executável do caso de teste de acordo. Certifique-se de fornecer o caminho correto do executável do caso de teste em cada arquivo `test.json` e de que o executável especificado seja executado corretamente.

Quando todos os dispositivos estiverem prontos para a execução de um caso de teste, o IDT lê os seguintes arquivos:

- O `test.json` para o caso de teste selecionado determina os processos a serem iniciados e as variáveis de ambiente a serem definidas.
- O `suite.json` para o pacote de teste determina as variáveis de ambiente a serem definidas.

O IDT inicia o processo executável de teste necessário com base nos comandos e argumentos especificados no arquivo `test.json` e passa as variáveis de ambiente necessárias para o processo.

Use o SDK do cliente do IDT

Os SDKs do IDT Client permitem simplificar a forma como a lógica de teste é escrita em seu executável de teste com comandos de API que podem ser usados para interagir com o IDT e seus dispositivos em teste. No momento, o IDT fornece os seguintes SDKs:

- SDK do cliente IDT para Python

- SDK do cliente IDT para Go
- SDK do cliente IDT para Java

Estes SDKs estão localizados na pasta *<device-tester-extract-location>*/sdks. Ao criar um novo executável de caso de teste, é preciso copiar o SDK que deseja usar para a pasta que contém o executável do caso de teste e referenciar o SDK em seu código. Esta seção fornece uma breve descrição dos comandos de API disponíveis que você pode usar nos executáveis do seu caso de teste.

Nesta seção

Interação do dispositivo

Os comandos a seguir permitem que a comunicação com o dispositivo em teste sem precisar implementar nenhuma interação adicional com o dispositivo e as funções de gerenciamento de conectividade.

`ExecuteOnDevice`

Permite que pacotes de teste executem comandos shell em um dispositivo compatível com conexões SSH ou Docker shell.

`CopyToDevice`

Permite que os pacotes de teste copiem um arquivo local da máquina host que executa o IDT para um local especificado em um dispositivo que suporte conexões SSH ou Docker shell.

`ReadFromDevice`

Permite que os pacotes de teste leiam a partir da porta serial de dispositivos compatíveis com conexões UART.

> **Note**
>
> Como o IDT não gerencia conexões diretas com dispositivos que são feitas usando as informações de acesso a dispositivos do contexto, recomendamos usar esses comandos

da API de interação de dispositivos em seus executáveis de casos de teste. No entanto, se esses comandos não atenderem aos requisitos do caso de teste, você poderá recuperar as informações de acesso ao dispositivo do contexto do IDT e usá-las para fazer uma conexão direta com o dispositivo do pacote de teste.
Para fazer uma conexão direta, recupere as informações nos campos `device.connectivity` e `resource.devices.connectivity` do dispositivo em teste e dos dispositivos de recursos, respectivamente. Para obter mais informações sobre como usar contexto do IDT, consulte Use o contexto do IDT.

Interação do IDT

Os comandos a seguir permitem que os conjuntos de teste se comuniquem com o IDT.

`PollForNotifications`

Permite que os pacotes de teste verifiquem as notificações do IDT.

`GetContextValue` e **`GetContextString`**

Permite que os pacotes de teste recuperem valores do contexto do IDT. Para ter mais informações, consulte Use o contexto do IDT.

`SendResult`

Permite que os pacotes de teste relatem os resultados dos casos de teste ao IDT. Este comando deve ser chamado no final de cada caso de teste em um pacote de teste.

Interação do host

O comando a seguir permite que seus pacotes de teste se comuniquem com a máquina host.

`PollForNotifications`

Permite que os pacotes de teste verifiquem as notificações do IDT.

`GetContextValue` e **`GetContextString`**

Permite que os pacotes de teste recuperem valores do contexto do IDT. Para ter mais informações, consulte Use o contexto do IDT.

`ExecuteOnHost`

Permite que os pacotes de teste executem comandos na máquina local e permite que o IDT gerencie o ciclo de vida do executável do caso de teste.

Habilitar comandos da CLI no IDT

O comando `run-suite` da CLI do IDT fornece várias opções que permitem que o executor de teste personalize a execução do teste. Para permitir que os executores de teste usem estas opções para executar seu pacote de teste personalizado, você implementa o suporte para a CLI do IDT. Se não implementar o suporte, os executores de teste ainda poderão executar testes, mas algumas opções da CLI não funcionarão corretamente. Para fornecer uma experiência ideal ao cliente, recomendamos que você implemente o suporte para os seguintes argumentos para o comando `run-suite` na CLI do IDT:

`timeout-multiplier`

Especifica um valor maior que 1,0 que será aplicado a todos os tempos limite durante a execução dos testes.

Os executores de teste podem usar esse argumento para aumentar o tempo limite dos casos de teste que desejam executar. Quando um executor de teste especifica esse argumento em seu comando `run-suite`, o IDT o usa para calcular o valor da variável de ambiente IDT_TEST_TIMEOUT e define o campo `config.timeoutMultiplier` no contexto do IDT. Para apoiar este argumento, você deve fazer o seguinte:

- Em vez de usar diretamente o valor de tempo limite do arquivo `test.json`, leia a variável de ambiente IDT_TEST_TIMEOUT para obter o valor de tempo limite calculado corretamente.
- Recupere o valor `config.timeoutMultiplier` do contexto do IDT e aplique-o a tempos limite de execução prolongados.

Para obter mais informações sobre como sair mais cedo por conta de eventos de tempo limite, consulte Especifique o comportamento de saída.

`stop-on-first-failure`

Especifica que o IDT deve parar de executar todos os testes se encontrar uma falha.

Quando um executor de teste especifica esse argumento no comando `run-suite`, o IDT interrompe a execução dos testes assim que encontrar uma falha. No entanto, se os casos de teste estiverem sendo executados em paralelo, isso poderá levar a resultados inesperados. Para

implementar o suporte, certifique-se de que, se o IDT encontrar esse evento, sua lógica de teste instrua todos os casos de teste em execução a parar, limpar recursos temporários e relatar o resultado do teste ao IDT. Para obter mais informações sobre sair antecipadamente em falhas, consulte Especifique o comportamento de saída.

`group-id` e `test-id`

Especifica que o IDT deve executar somente os grupos de teste ou casos de teste selecionados.

Os executores de teste podem usar esses argumentos com os comandos `run-suite` para especificar o seguinte comportamento de execução do teste:

- Execute todos os testes dentro dos grupos de teste especificados.
- Execute uma seleção de testes de dentro de um grupo de teste especificado.

Para apoiar esses argumentos, a máquina de estados do seu pacote de teste deve incluir um conjunto específico de estados `RunTask` e `Choice` em sua máquina de estado. Se não estiver usando uma máquina de estado personalizada, a máquina de estado IDT padrão incluirá os estados necessários e você não precisará realizar nenhuma ação adicional. No entanto, se estiver usando uma máquina de estado personalizada, use Exemplo de máquina de estado: execute grupos de teste selecionados pelo usuário como amostra para adicionar os estados necessários à sua máquina de estado.

Para obter mais informações sobre os comandos da CLI no IDT, consulte Depure e execute pacotes de teste personalizados.

Gravar logs de eventos

Enquanto o teste está sendo executado, você envia dados para `stdout` e `stderr` e grava logs de eventos e mensagens de erro no console. Para obter informações sobre o formato das mensagens do console, consulte Formato de mensagem do console.

Quando o IDT terminar de executar o pacote de teste, essas informações também estarão disponíveis no arquivo `test_manager.log` localizado na pasta *`<devicetester-extract-location>`*`/results/`*`<execution-id>`*`/logs`.

É possível configurar cada caso de teste para gravar os logs de sua execução de teste, incluindo logs do dispositivo em teste, no arquivo *`<group-id>`*`_`*`<test-id>`* localizado na pasta *`<device-tester-extract-location>`*`/results/`*`execution-id`*`/logs`. Para fazer isso, recupere o caminho para o arquivo de log do contexto do IDT com a consulta `testData.logFilePath`, crie

um arquivo nesse caminho e grave o conteúdo deseja nele. O IDT atualiza automaticamente o caminho com base no caso de teste que está sendo executado. Se você optar por não criar o arquivo de log para um caso de teste, nenhum arquivo será gerado para esse caso de teste.

É possível configurar seu executável de texto para criar arquivos de log adicionais, conforme necessário, na pasta *<device-tester-extract-location>*/logs. É recomendado especificar prefixos exclusivos para nomes de arquivos de log para que seus arquivos não sejam substituídos.

Relatar resultados ao IDT

O IDT grava os resultados do teste nos arquivos awsiotdevicetester_report.xml e *suite-name*_report.xml. Estes arquivos de relatório estão localizados em *<device-tester-extract-location>*/results/*<execution-id>*/. Ambos os relatórios capturam os resultados da execução do pacote de teste. Para obter mais informações sobre os esquemas que o IDT usa para esses relatórios, consulte Analise os resultados e logs dos testes do IDT

Para preencher o conteúdo do arquivo *suite-name*_report.xml, o comando SendResult deve ser usado para relatar os resultados do teste ao IDT antes da conclusão da execução do teste. Se o IDT não conseguir localizar os resultados de um teste, ele emitirá um erro para o caso de teste. O seguinte trecho em Python mostra os comandos para enviar um resultado de teste para o IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Se não reportar os resultados por meio da API, o IDT procurará os resultados do teste na pasta de artefatos de teste. O caminho para essa pasta é armazenado no campo testData.testArtifactsPath no contexto do IDT. Nesta pasta, o IDT usa o primeiro arquivo XML classificado em ordem alfabética localizado como resultado do teste.

Se sua lógica de teste produzir resultados XML JUnit, será possível gravar os resultados do teste em um arquivo XML na pasta de artefatos para fornecer os resultados diretamente ao IDT em vez de analisá-los e depois usar a API para enviá-los ao IDT.

Se usar este método, a lógica de teste deverá resumir com precisão os resultados do teste e formatar o arquivo de resultados no mesmo formato do arquivo *suite-name*_report.xml. O IDT não realiza nenhuma validação dos dados fornecidos, com as seguintes exceções:

- O IDT ignora todas as propriedades da tag testsuites. Em vez disso, ele calcula as propriedades da tag a partir dos resultados de outros grupos de teste relatados.
- Pelo menos uma tag testsuite deve existir nos testsuites.

Como o IDT usa a mesma pasta de artefatos para todos os casos de teste e não exclui os arquivos de resultados entre as execuções de teste, esse método também pode gerar relatórios incorretos, caso o IDT leia o arquivo incorreto. É recomendado o uso do mesmo nome para o arquivo de resultados XML gerado em todos os casos de teste para sobrescrever os resultados de cada caso de teste e garantir que os resultados corretos estejam disponíveis para o IDT usar. Embora seja possível usar uma abordagem mista para gerar relatórios em pacotes de teste, ou seja, usar um arquivo de resultados XML para alguns casos de teste e enviar resultados por meio da API em outros casos, não recomendamos essa abordagem.

Especifique o comportamento de saída

Configure seu executável de texto para sempre sair com um código de saída 0, mesmo se um caso de teste relatar uma falha ou um resultado de erro. Use códigos de saída diferentes de zero somente para indicar que um caso de teste não foi executado ou caso o executável do caso de teste não tenha comunicado nenhum resultado ao IDT. Quando o IDT recebe um código de saída diferente de zero, ele marca que o caso de teste se deparou com um erro que o impediu de ser executado.

O IDT pode solicitar ou esperar a interrupção da execução de um caso de teste antes de concluir os eventos a seguir. Use estas informações para configurar o executável do caso de teste para detectar cada um desses eventos do caso de teste:

Timeout (Tempo limite)

Ocorre quando um caso de teste é executado por mais tempo do que o valor de tempo limite especificado no arquivo `test.json`. Se o executor do teste usou o argumento `timeout-multiplier` para especificar um multiplicador de tempo limite, o IDT calcula o valor do tempo limite com o multiplicador.

Para detectar este evento, use a variável de ambiente IDT_TEST_TIMEOUT. Quando um executor de teste inicializa um teste, o IDT define o valor da variável de ambiente IDT_TEST_TIMEOUT como o valor de tempo limite calculado (em segundos) e passa a variável para o executável do caso de teste. É possível ler o valor da variável para definir um cronômetro apropriado.

Interromper

Ocorre quando o executor do teste interrompe o IDT. Por exemplo, ao pressionar Ctrl+C.

Como os terminais propagam sinais para todos os processos secundários, você pode simplesmente configurar um manipulador de sinais em seus casos de teste para detectar sinais de interrupção.

Como alternativa, é possível sondar periodicamente a API para verificar o valor da `CancellationRequested` booleana na resposta da API `PollForNotifications`. Quando o IDT recebe um sinal de interrupção, ele define o valor do `CancellationRequested` booleano como `true`.

Interrompa na primeira falha

Ocorre quando um caso de teste executado paralelamente ao caso de teste atual falha e o executor de teste usa o argumento `stop-on-first-failure` para especificar que o IDT deve interromper ao se deparar com uma falha.

Para detectar esse evento, é possível pesquisar periodicamente a API para verificar o valor `CancellationRequested` do booleano na resposta da API `PollForNotifications`. Quando o IDT encontra uma falha e é configurado para interromper a primeira falha, ele define o valor do `CancellationRequested` booleano como `true`.

Quando um desses eventos ocorre, o IDT espera cinco minutos para que os casos de teste em execução no momento concluam a execução. Se todos os casos de teste em execução não saírem em cinco minutos, o IDT forçará a interrupção de cada um dos processos deles. Se o IDT não tiver recebido os resultados do teste antes da conclusão dos processos, ele marcará os casos de teste como expirados. Como prática recomendada, os seus casos de teste devem executar as seguintes ações quando encontrarem um dos eventos:

1. Interrompa a execução da lógica de teste normal.
2. Limpar todos os recursos temporários, como artefatos de teste no dispositivo em teste.
3. Relate um resultado de teste ao IDT, como uma falha ou erro no teste.
4. Sair.

Use o contexto do IDT

Quando o IDT executa um pacote de teste, o pacote de teste pode acessar um conjunto de dados que podem ser usados para determinar como cada teste é executado. Estes dados são chamados de contexto do IDT. Por exemplo, a configuração de dados do usuário fornecida pelos executores de teste em um arquivo `userdata.json` é disponibilizada para pacotes de teste no contexto do IDT.

O contexto do IDT pode ser considerado um documento JSON somente para leitura. Os pacotes de teste podem recuperar e gravar dados no contexto usando tipos de dados JSON padrão, como objetos, matrizes, números e assim por diante.

Esquema de contexto

O contexto do IDT usa o formato a seguir:

```
{
    "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`

As informações do arquivo `config.json`. O campo `config` também contém os seguintes campos adicionais:

`config.timeoutMultiplier`

O multiplicador para qualquer valor de tempo limite usado pelo pacote de teste. Este valor é especificado pelo executor de teste da CLI do IDT. O valor padrão é 1.

`config.idRootPath`

Este valor é um espaço reservado para o valor absoluto do caminho do IDT durante a configuração do arquivo `userdata.json`. Isto é usado pelos comandos de compilação e atualização.

`device`

As informações sobre o dispositivo selecionado para a execução do teste. Estas informações são equivalentes à matriz `devices` definida no arquivo `device.json` do dispositivo selecionado.

`devicePool`

Informações sobre o grupo de dispositivos selecionado para a execução do teste. Estas informações são equivalentes ao elemento de matriz do grupo de dispositivos de nível superior definido no arquivo `device.json` do grupo de dispositivos selecionado.

`resource`

As informações sobre dispositivos de recursos do arquivo `resource.json`.

`resource.devices`

Estas informações são equivalentes à matriz `devices` definida no arquivo `resource.json`. Cada elemento `devices` inclui o seguinte campo adicional:

`resource.device.name`

O nome do dispositivo de recurso. Este valor é definido como o valor `requiredResource.name` no arquivo `test.json`.

`testData.awsCredentials`

As AWS credenciais usadas pelo teste para se conectar à AWS nuvem. Estas informações são obtidas do arquivo `config.json`.

`testData.logFilePath`

O caminho para o arquivo de log no qual o caso de teste grava mensagens de log. O pacote de teste criará este arquivo se ele não existir.

`userData`

As informações fornecidas pelo executor de teste no arquivo `userdata.json`.

Acesse dados no contexto

É possível consultar o contexto usando a notação JSONPath dos seus arquivos de configuração e do seu executável de texto com as APIs `GetContextValue` e `GetContextString`. A sintaxe das strings JSONPath para acessar o contexto do IDT varia da seguinte forma:

- Em `suite.json` e `test.json`, é usado `{{`*`query`*`}}`. Ou seja, não use o elemento raiz `$.` para iniciar sua expressão.
- Em `statemachine.json`, é usado `{{$.`*`query`*`}}`.
- Nos comandos da API, é usado *`query`* ou `{{$.`*`query`*`}}`, dependendo do comando. Para obter mais informações, consulte a documentação em linha nos SDKs.

A tabela a seguir descreve os operadores em uma expressão JSONPath típica de foobar:

Operador	Descrição
`$`	O elemento raiz. Como o valor de contexto de nível superior do IDT é um objeto, normalmente você usará `$.` para iniciar suas consultas.
`.childName`	Acessa o elemento filho com o nome `childName` de um objeto. Se aplicado a uma matriz, produz uma nova matriz com esse operador aplicado a cada elemento. O nome do elemento diferencia maiúsculas e minúsculas. Por exemplo, a consulta para acessar o valor `awsRegion` no objeto `config` é `$.config.awsRegion` .
`[start:end]`	Filtra elementos de uma matriz, recuperando itens que começam no índice `start` e vão até o índice `end`, ambos inclusive.

Operador	Descrição
`[index1, index2, ... , indexN]`	Filtra elementos de uma matriz, recuperando itens somente dos índices especificados.
`[?(expr)]`	Filtra elementos de uma matriz usando a expressão `expr`. Esta expressão que deve ser avaliada para um valor booleano.

Para criar expressões de filtro, use os seguintes elementos:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

Nesta sintaxe:

- `jsonpath` é um JSONPath que usa a sintaxe JSON padrão.
- `value` é qualquer valor personalizado que usa a sintaxe JSON padrão.
- `operator` é um dos seguintes operadores:
  - < (Menor que)
  - <= (Menor ou igual a)
  - == (Igual a)

    Se o JSONPath ou o valor em sua expressão for um valor de matriz, booleano ou objeto, este será o único operador binário compatível que você poderá usar.
  - >= (Maior ou igual a)
  - > (Maior que)
  - =~ (Correspondência de expressão regular). Para usar este operador em uma expressão de filtro, o JSONPath ou valor no lado esquerdo da expressão deve ser avaliado como uma string e o lado direito deve ser um valor padrão que siga a sintaxe RE2.

É possível usar consultas JSONPath no formato {{*query*}} como strings de espaço reservado nos campos `args` e `environmentVariables` nos arquivos `test.json`, e nos campos `environmentVariables` nos arquivos `suite.json`. O IDT realiza uma pesquisa de contexto e preenche os campos com o valor avaliado da consulta. Por exemplo, no arquivo `suite.json`, é possível usar strings de espaço reservado para especificar valores de variáveis de ambiente que

mudam com cada caso de teste e o IDT preencherá as variáveis de ambiente com o valor correto para cada caso de teste. No entanto, ao usar strings de caracteres de espaço reservado em arquivos `test.json` e `suite.json`, as seguintes considerações se aplicam às suas consultas:

- Cada ocorrência da chave `devicePool` em sua consulta deve estar em letras minúsculas. Ou seja, em vez disso, use `devicepool`.
- Para as matrizes, somente matrizes de strings podem ser usadas. Além disso, as matrizes usam um formato `item1, item2,...,itemN` não padrão. Se a matriz contiver somente um elemento, ela será serializada como `item`, tornando-a indistinguível de um campo de strings.
- Não é possível usar espaços reservados para recuperar objetos do contexto.

Por conta dessas considerações, é recomendado, sempre que possível, usar a API para acessar o contexto em sua lógica de teste em vez de strings de espaço reservado em arquivos `test.json` e `suite.json`. No entanto, em alguns casos, pode ser mais conveniente usar espaços reservados JSONPath para recuperar strings únicas para definir como variáveis de ambiente.

Definir configurações para os executores de teste

Para executar pacotes de testes personalizados, os executores de teste devem definir suas configurações com base no pacote de teste que desejam executar. As configurações são especificadas com base nos modelos de arquivo de configuração localizados na pasta *`<device-tester-extract-location>`*`/configs/`. Se necessário, os executores de teste também devem configurar AWS as credenciais que a IDT usará para se conectar à nuvem. AWS

Como escritor de teste, será necessário configurar esses arquivos para depurar seu pacote de teste. É preciso fornecer instruções aos executores de teste para que eles possam definir as seguintes configurações conforme necessário para executar seus pacotes de testes.

Configurar device.json

O arquivo `device.json` contém informações sobre os dispositivos em que os testes são executados (por exemplo, endereço IP, informações de login, sistema operacional e arquitetura de CPU).

Os executores de teste podem fornecer essas informações usando o seguinte arquivo `device.json` de modelo localizado na pasta *`<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 os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID alfanumérico definido pelo usuário que identifica uma coleção de dispositivos chamada de grupo de dispositivos. Os dispositivos que pertencem a um grupo devem ter hardware idêntico. Quando um conjunto de testes é executado, os dispositivos do grupo são usados para paralelizar a carga de trabalho. Vários dispositivos são usados para executar testes diferentes.

`sku`

Um valor alfanumérico que identifica exclusivamente o dispositivo em teste. A SKU é usada para rastrear os dispositivos qualificados.

> **Note**
>
> Se você quiser listar sua placa no Catálogo de dispositivos AWS parceiros, o SKU especificado aqui deve corresponder ao SKU que você usa no processo de listagem.

`features`

Opcional. Uma matriz que contém recursos compatíveis com o dispositivo. Os recursos do dispositivo são valores definidos pelo usuário que você configura em seu pacote de teste. É preciso fornecer informações aos executores de teste sobre os nomes e valores dos recursos a serem incluídos no arquivo `device.json`. Por exemplo, se você quiser testar um dispositivo que funciona como um servidor MQTT para outros dispositivos, você pode configurar sua lógica de teste para validar níveis específicos suportados para um recurso chamado `MQTT_QoS`. Os executores de teste fornecem este nome de recurso e definem o valor do recurso para os níveis de QoS compatíveis com o dispositivo. É possível recuperar as informações fornecidas do contexto do IDT com a consulta `devicePool.features` ou do contexto da máquina de estado com a consulta `pool.features`.

`features.name`

O nome do recurso.

`features.value`

Os valores dos recursos compatíveis.

`features.configs`

Definições de configuração, se necessárias, para o recurso.

`features.config.name`

O nome do ajuste de configurações.

`features.config.value`

Os valores de configuração compatíveis.

`devices`

Uma matriz de dispositivos no grupo a serem testados. Pelo menos um dispositivo é necessário.

`devices.id`

Um identificador exclusivo, definido pelo usuário, para o dispositivo que está sendo testado.

`devices.pairedResource`

Um identificador exclusivo, definido pelo usuário, para um dispositivo de recursos. Este valor é necessário quando você testa dispositivos usando o protocolo de conectividade `no-op`.

`connectivity.protocol`

O protocolo de comunicação usado para se comunicar com esse dispositivo. Cada dispositivo em um grupo deve usar o mesmo protocolo.

No momento, os únicos valores compatíveis são `ssh` e `uart` para dispositivos físicos, `docker` para contêineres do Docker e `no-op` para dispositivos que não tenham uma conexão direta com a máquina host do IDT, mas precisam de um dispositivo de recursos como middleware físico para se comunicar com a máquina host.

Para dispositivos não operacionais, você configura o ID do dispositivo de recurso em `devices.pairedResource`. Também é preciso especificar esse ID no arquivo `resource.json`. O dispositivo emparelhado deve ser um dispositivo fisicamente emparelhado com o dispositivo em teste. Depois que o IDT identificar e se conectar ao dispositivo de recursos emparelhado, o IDT não se conectará a outros dispositivos de recursos de acordo com os recursos descritos no arquivo `test.json`.

`connectivity.ip`

O endereço IP do dispositivo que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.port`

Opcional. O número da porta usada nas conexões SSH.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.publicKeyPath`

Opcional. O caminho completo para a chave pública usada para autenticar conexões com o dispositivo em teste. Ao especificar o `publicKeyPath`, o IDT valida a chave pública do dispositivo ao estabelecer uma conexão SSH com o dispositivo em teste. Se este valor não for especificado, o IDT cria uma conexão SSH, mas não valida a chave pública do dispositivo.

É altamente recomendável especificar o caminho para a chave pública e use um método seguro para buscar essa chave pública. Para clientes SSH padrão baseados em linha de comando, a chave pública é fornecida no arquivo `known_hosts`. Se especificar um arquivo de chave pública separado, esse arquivo deverá usar o mesmo formato do arquivo `known_hosts` , ou seja, `ip-address key-type public-key`.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`

- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`connectivity.auth.credentials.user`

O nome de usuário para fazer login no dispositivo que está sendo testado.

`connectivity.serialPort`

Opcional. A porta serial à qual o dispositivo está conectado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `uart`.

`connectivity.containerId`

O ID do contêiner ou o nome do contêiner do Docker que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `docker`.

`connectivity.containerUser`

Opcional. O nome de usuário para o usuário dentro do contêiner. O valor padrão é o usuário fornecido no Dockerfile.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `docker`.

> **Note**
>
> Para verificar se os executores de teste configuram a conexão incorreta do dispositivo para um teste, você pode recuperar o `pool.Devices[0].Connectivity.Protocol` do contexto da máquina de estado e compará-la com o valor esperado em um estado `Choice`. Se um protocolo incorreto for usado, imprima uma mensagem usando o estado `LogMessage` e faça a transição para o estado `Fail`.
> Como alternativa, é possível usar o código de tratamento de erros para relatar uma falha no teste para tipos de dispositivos incorretos.

(Opcional) Configurar userdata.json

O arquivo `userdata.json` contém qualquer informação adicional exigida por um pacote de teste, mas não está especificada no arquivo `device.json`. O formato deste arquivo depende do arquivo `userdata_scheme.json` definido no pacote de teste. Se for um redator de testes, certifique-se de fornecer essas informações aos usuários que executarão os pacotes de testes escritos.

(Opcional) Configurar resource.json

O arquivo `resource.json` contém informações sobre todos os dispositivos que serão usados como dispositivos de recursos. Os dispositivos de recursos são dispositivos necessários para testar determinados recursos de um dispositivo em teste. Por exemplo, para testar a capacidade Bluetooth de um dispositivo, é possível usar um dispositivo de recurso para testar se seu dispositivo consegue se conectar com êxito. Os dispositivos de recursos são opcionais e pode exigir quantos dispositivos de recursos precisar. Como redator de teste, é possível usar o arquivo test.json para definir os recursos do dispositivo de recursos necessários para um teste. Em seguida, os executores de teste usam o arquivo `resource.json` para fornecer um pool de dispositivos de recursos com os recursos necessários. Certifique-se de fornecer essas informações aos usuários que executarão os pacotes de testes escritos.

Os executores de teste podem fornecer essas informações usando o seguinte arquivo `resource.json` de modelo localizado na pasta *`<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 os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID alfanumérico definido pelo usuário que identifica uma coleção de dispositivos chamada de grupo de dispositivos. Os dispositivos que pertencem a um grupo devem ter hardware idêntico. Quando um conjunto de testes é executado, os dispositivos do grupo são usados para paralelizar a carga de trabalho. Vários dispositivos são usados para executar testes diferentes.

`features`

Opcional. Uma matriz que contém recursos compatíveis com o dispositivo. As informações necessárias neste campo são definidas nos arquivos test.json no pacote de teste e determinam quais testes devem ser executados e como executá-los. Se o pacote de teste não exigir nenhum recurso, este campo não será obrigatório.

`features.name`

O nome do recurso.

`features.version`

A versão do recurso.

`features.jobSlots`

Configuração para indicar quantos testes podem usar o dispositivo simultaneamente. O valor padrão é 1.

`devices`

Uma matriz de dispositivos no grupo a serem testados. Pelo menos um dispositivo é necessário.

`devices.id`

Um identificador exclusivo, definido pelo usuário, para o dispositivo que está sendo testado.

`connectivity.protocol`

O protocolo de comunicação usado para se comunicar com esse dispositivo. Cada dispositivo em um grupo deve usar o mesmo protocolo.

No momento, os únicos valores compatíveis são `ssh` e `uart` para dispositivos físicos e `docker` para contêineres do Docker.

`connectivity.ip`

O endereço IP do dispositivo que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.port`

Opcional. O número da porta usada nas conexões SSH.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.publicKeyPath`

Opcional. O caminho completo para a chave pública usada para autenticar conexões com o dispositivo em teste. Ao especificar o `publicKeyPath`, o IDT valida a chave pública do dispositivo ao estabelecer uma conexão SSH com o dispositivo em teste. Se este valor não for especificado, o IDT cria uma conexão SSH, mas não valida a chave pública do dispositivo.

É altamente recomendável especificar o caminho para a chave pública e use um método seguro para buscar essa chave pública. Para clientes SSH padrão baseados em linha de comando, a chave pública é fornecida no arquivo `known_hosts`. Se especificar um arquivo de chave pública separado, esse arquivo deverá usar o mesmo formato do arquivo `known_hosts` , ou seja, `ip-address key-type public-key`.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`connectivity.auth.credentials.user`

O nome de usuário para fazer login no dispositivo que está sendo testado.

`connectivity.serialPort`

Opcional. A porta serial à qual o dispositivo está conectado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `uart`.

`connectivity.containerId`

O ID do contêiner ou o nome do contêiner do Docker que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `docker`.

`connectivity.containerUser`

Opcional. O nome de usuário para o usuário dentro do contêiner. O valor padrão é o usuário fornecido no Dockerfile.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `docker`.

(Opcional) Configurar config.json

O arquivo `config.json` contém as informações de configuração do IDT. Normalmente, os executores de teste não precisarão modificar esse arquivo, exceto para fornecer suas credenciais de AWS usuário para o IDT e, opcionalmente, para uma região. AWS Se AWS as credenciais com as permissões necessárias forem fornecidas, AWS IoT Device Tester coleta e envia métricas de uso para o. AWS Esse atributo é opcional e usado para aprimorar a funcionalidade do IDT. Para ter mais informações, consulte Métricas de uso do IDT.

Os executores de teste podem configurar suas AWS credenciais de uma das seguintes formas:

- Arquivo de credenciais

  O IDT usa o mesmo arquivo de credenciais que a AWS CLI. Para obter mais informações, consulte Arquivos de configuração e credenciais.

  O local do arquivo de credenciais varia de acordo com o sistema operacional que você está usando:

  - macOS, Linux: `~/.aws/credentials`
  - Windows: `C:\Users\UserName\.aws\credentials`
- Variáveis de ambiente

  As variáveis de ambiente são variáveis mantidas pelo sistema operacional e usadas pelos comandos do sistema. As variáveis definidas durante uma sessão SSH não estão disponíveis após o encerramento da sessão. O IDT pode usar as variáveis de ambiente `AWS_ACCESS_KEY_ID` e `AWS_SECRET_ACCESS_KEY` para armazenar suas credenciais da AWS .

  Para definir essas variáveis no Linux, macOS ou Unix, use :export

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

  Para definir essas variáveis no Windows, use :set

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Para configurar AWS as credenciais para o IDT, os executores de teste editam a `auth` seção no `config.json` arquivo localizado na pasta. *<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 os campos que contêm valores são necessários, conforme descrito aqui:

> **Note**
>
> Todos os caminhos nesse arquivo são definidos em relação ao *< device-tester-extract-location >*.

`log.location`

O caminho para a pasta de registros em *< device-tester-extract-location >*.

`configFiles.root`

O caminho para a pasta que contém os arquivos de configuração.

`configFiles.device`

O caminho para a função `device.json`.

`testPath`

O caminho para a pasta que contém pacotes de teste.

`reportPath`

O caminho para a pasta que conterá os resultados do teste depois que o IDT executar um pacote de teste.

`awsRegion`

Opcional. A AWS região que as suítes de teste usarão. Se não for definida, os pacotes de teste usarão a região padrão especificada em cada pacote de teste.

`auth.method`

O método que o IDT usa para recuperar credenciais AWS . Os valores aceitos são `file` recuperar credenciais de um arquivo de credenciais e recuperar credenciais `environment` usando variáveis de ambiente.

`auth.credentials.profile`

O perfil de credenciais a ser usado no arquivo de credenciais. Essa propriedade será aplicada somente se `auth.method` estiver definido como `file`.

Depure e execute pacotes de teste personalizados

Depois que a configuração necessária for definida, o IDT poderá executar seu pacote de teste. O runtime do pacote de teste completo depende do hardware e da composição do pacote de teste. Como referência, leva aproximadamente 30 minutos para concluir o pacote de teste do FreeRTOS completo em um Raspberry Pi 3B.

À medida que seu pacote de teste é escrito, é possível usar o IDT para executar o pacote de teste no modo de depuração para verificar seu código antes de executá-lo ou fornecê-lo aos executores de teste.

Executar o IDT no modo de depuração

Como os pacotes de teste dependem do IDT para interagir com dispositivos, fornecer o contexto e receber resultados, não é possível simplesmente depurar seus pacotes de teste em um IDE sem qualquer interação com o IDT. Para fazer isso, a CLI do IDT fornece o comando `debug-test-suite` que permite executar o IDT no modo de depuração. Execute o seguinte comando para visualizar as opções disponíveis para `debug-test-suite`:

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

Ao executar o IDT no modo de depuração, o IDT na verdade não inicia o pacote de teste nem executa o orquestrador de testes. Em vez disso, ele interage com seu IDE para responder às solicitações feitas do pacote de teste em execução no IDE e imprime os logs no console. O IDT não atinge o tempo limite e espera para sair até ser interrompido manualmente. No modo de depuração, o IDT também não executa o orquestrador de testes e não gera nenhum arquivo de relatório. Para depurar seu pacote de teste, você deve usar seu IDE para fornecer algumas informações que o IDT normalmente obtém dos arquivos de configuração. Certifique-se de fornecer as seguintes informações:

- Variáveis de ambiente e argumentos para cada teste. O IDT não lerá essas informações de `test.json` ou `suite.json`.
- Argumentos para selecionar os dispositivos de recursos. O IDT não lerá essas informações de `test.json`.

Para depurar seus pacotes de teste, conclua as seguintes etapas:

1. Crie os arquivos de ajuste de configuração necessários para executar o pacote de teste. Por exemplo, se seu pacote de teste exigir o `device.json`, `resource.json` e `user data.json` certifique-se de configurar todos eles conforme necessário.
2. Execute o comando a seguir para colocar o IDT no modo de depuração e selecionar todos os dispositivos necessários para executar o teste.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   Depois de executar esse comando, o IDT aguarda as solicitações do pacote de teste e responde a elas. O IDT também gera as variáveis de ambiente necessárias para o processo de caso do SDK do cliente de IDT.
3. No seu IDE, use a configuração `run` ou `debug` para fazer o seguinte:
   a. Defina os valores das variáveis de ambiente geradas pelo IDT.
   b. Defina o valor de qualquer variável de ambiente ou argumento que você especificou em seu arquivo `test.json` e `suite.json`.
   c. Definir pontos de interrupção, conforme necessário.
4. Execute o pacote de teste em seu IDE.

É possível depurar e executar novamente o pacote de teste quantas vezes for necessário. O IDT não atinge o tempo limite no modo de depuração.

5. Depois de concluir a depuração, interrompa o IDT para sair do modo de depuração.

Comandos da CLI do IDT para executar testes

As seções a seguir descrevem os comandos da CLI do IDT.

IDT v4.0.0

`help`

Lista as informações sobre o comando especificado.

`list-groups`

Lista os grupos em um determinado conjunto de teste.

`list-suites`

Lista os conjuntos de teste disponíveis.

`list-supported-products`

Lista os produtos compatíveis para a sua versão do IDT (neste caso, versões do IDT) e versões do pacote de teste de qualificação do FreeRTOS disponíveis para a versão atual do IDT.

`list-test-cases`

Lista os casos de teste em um grupo de teste. A seguinte opção é compatível:

- `group-id`. O grupo de teste a ser pesquisado. Esta opção é necessária e deve especificar um único grupo.

`run-suite`

Executa um conjunto de testes em um grupo de dispositivos. Algumas opções comumente usadas a seguir:

- `suite-id`. A versão do pacote de teste a ser executada. Se não for especificado, o IDT usará a versão mais recente na pasta `tests`.
- `group-id`. Os grupos de teste a serem executados, como uma lista separada por vírgulas. Se não for especificado, o IDT executa todos os grupos de teste no conjunto de testes.

- `test-id`. Os casos de teste a serem executados, como uma lista separada por vírgulas. Quando especificado, `group-id` deve especificar um único grupo.
- `pool-id`. O grupo de dispositivos a ser testado. É preciso especificar um grupo se houver vários grupos de dispositivos definidos no arquivo `device.json`.
- `timeout-multiplier`. Configura o IDT para modificar o tempo limite de execução do teste especificado no arquivo `test.json` para um teste com um multiplicador definido pelo usuário.
- `stop-on-first-failure`. Configura o IDT de modo a interromper a execução na primeira falha. Essa opção deve ser usada com para depurar os grupos de teste especificados `group-id`.
- `userdata`. Define o arquivo que contém as informações de dados do usuário necessárias para executar o pacote de teste. Isto é necessário somente se `userdataRequired` estiver definido como verdadeiro no arquivo `suite.json` do pacote de teste.

Para obter mais informações sobre as opções `run-suite`, use a opção `help`:

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

`debug-test-suite`

Execute o pacote de teste no modo de depuração. Para ter mais informações, consulte Executar o IDT no modo de depuração.

Analise os resultados e logs dos testes do IDT

Esta seção descreve o formato no qual o IDT gera logs do console e relatórios de teste.

Formato de mensagem do console

AWS IoT Device Tester usa um formato padrão para imprimir mensagens no console quando ele inicia uma suíte de testes. O trecho a seguir mostra um exemplo de uma mensagem de console gerada pelo IDT.

```
[INFO] [2000-01-02 03:04:05]: Using suite: MyTestSuite_1.0.0
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

A maioria das mensagens detalhadas do console consiste nos seguintes campos:

`time`

Um carimbo de data/hora ISO 8601 completo para o evento registrado.

`level`

O nível da mensagem para o evento registrado. Normalmente, o nível da mensagem registrada é um `info`, `warn` ou `error`. O IDT emite uma mensagem `fatal` ou `panic` se encontrar um evento esperado que faça com que ele saia antecipadamente.

`msg`

A mensagem registrada.

`executionId`

Uma string de ID exclusiva para o processo atual do IDT. Este ID é usado para diferenciar entre execuções individuais de IDT.

As mensagens do console geradas de um pacote de teste fornecem informações adicionais sobre o dispositivo em teste e o pacote de teste, o grupo de teste e os casos de teste que o IDT executa. O trecho a seguir mostra um exemplo de uma mensagem de console gerada de um pacote de teste.

```
[INFO] [2000-01-02 03:04:05]: Hello world! suiteId=MyTestSuitegroupId=myTestGroup
 testCaseId=myTestCase deviceId=my-
deviceexecutionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

A parte específica do pacote de teste da mensagem do console contém os seguintes campos:

`suiteId`

O nome do pacote de teste em execução no momento.

`groupId`

O ID do grupo de teste em execução no momento.

`testCaseId`

O ID do caso de teste em execução no momento.

`deviceId`

Um ID do dispositivo em teste que o caso de teste atual está usando.

O resumo do teste contém informações sobre o pacote de teste, os resultados de cada grupo executado e os locais dos registros e arquivos de logs gerados. O exemplo a seguir mostra uma mensagem de resume do teste.

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

AWS IoT Device Tester esquema de relatório

`awsiotdevicetester_report.xml` é um relatório assinado que contém as seguintes informações:

- A versão IDT.
- A versão do pacote de teste.
- A assinatura do relatório e a chave usada para assinar o relatório.
- A SKU do dispositivo e o nome de grupo do dispositivo especificados no arquivo `device.json`.
- A versão do produto e os recursos do dispositivo que foram testados.
- O resumo agregado dos resultados de teste. Estas informações são as mesmas contidas no arquivo *`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>
```

O arquivo `awsiotdevicetester_report.xml` contém uma tag `<awsproduct>` com informações sobre o produto que está sendo testado e os recursos do produto que foram validados após a execução de um pacote de testes.

recursos usados na tag **`<awsproduct>`**

`name`

O nome do produto testado.

`version`

A versão do produto testado.

`features`

Os recursos validados. Os recursos marcados como `required` são necessários para que o pacote de teste valide o dispositivo. O trecho a seguir mostra como essas informações aparecem no arquivo `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Os recursos marcados como não `optional` são necessários para validação. Os seguintes trechos mostram recursos opcionais.

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Esquema do relatório do pacote de teste

O relatório *`suite-name`*`_Result.xml` está no formato JUnit XML. Você pode integrá-lo em plataformas de integração e implantação como Jenkins, Bamboo e assim por diante. O relatório contém um resumo agregado dos resultados de teste.

```
<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>
```

A seção de relatório tanto em `awsiotdevicetester_report.xml` como em *`suite-name`*`_report.xml` lista os testes que foram executados e os resultados.

A primeira tag XML `<testsuites>` contém o resumo da execução do teste. Por exemplo: .

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

recursos usados na tag **`<testsuites>`**

`name`

O nome do conjunto de testes.

`time`

O tempo, em segundos, necessário para executar o pacote de teste.

`tests`

O número de testes executados.

`failures`

O número de testes que foram executados, mas não foram aprovados.

`errors`

O número de testes que não puderam ser executados pelo IDT.

`disabled`

Esse recurso não é usado e pode ser ignorado.

Se houver falhas de teste ou erros, você poderá identificar o teste com falha analisando as tags XML `<testsuites>`. As tags XML `<testsuite>` dentro da tag `<testsuites>` mostram o resumo do resultado do teste para um grupo de testes. Por exemplo: .

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

O formato é semelhante à tag `<testsuites>`, mas com um recurso `skipped` que não é usado e pode ser ignorado. Dentro de cada tag XML `<testsuite>`, há tags `<testcase>` para cada teste executado para um grupo de testes. Por exemplo: .

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

recursos usados na tag **`<testcase>`**

`name`

O nome do teste.

`attempts`

O número de vezes que o IDT executou o caso de teste.

Quando um teste falha ou ocorre um erro, as tags `<failure>` ou `<error>` são adicionadas à tag `<testcase>` com informações para a solução de problemas. Por exemplo: .

```
<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 do IDT

Se você fornecer AWS credenciais com as permissões necessárias, AWS IoT Device Tester coletará e enviará métricas de uso para. AWS Este é um recurso opcional e é usado para melhorar a funcionalidade do IDT. O IDT coleta informações como as seguintes:

- O ID AWS da conta usado para executar o IDT
- Os comandos da CLI do IDT usados para executar testes
- O pacote de teste que é executado

- As suítes de teste na pasta *< device-tester-extract-location >*
- O número de dispositivos configurados no grupo de dispositivos
- Nomes de casos de teste e tempos de execução
- Informações do resultado do teste, como se os testes foram aprovados, falharam, encontraram erros ou foram ignorados
- recursos testados do produto
- Comportamento de saída do IDT, como saídas inesperadas ou antecipadas

Todas as informações enviadas pelo IDT também são registradas em um arquivo `metrics.log` na pasta *<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Você pode visualizar o arquivo de log para ver as informações que foram coletadas durante a execução de um teste. Este arquivo é gerado somente se optar por coletar métricas de uso.

Para desativar a coleta de métricas, não é necessário tomar nenhuma outra medida. Simplesmente não armazene suas AWS credenciais e, se você tiver AWS credenciais armazenadas, não configure o `config.json` arquivo para acessá-las.

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

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

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

Para conceder acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos em AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Criação de um conjunto de permissões no Guia do usuário do AWS IAM Identity Center .
- Usuários gerenciados no IAM com provedor de identidades:

  Crie um perfil para a federação de identidades. Siga as instruções em Criar um perfil para um provedor de identidades de terceiros (federação) no Guia do usuário do IAM.
- Usuários do IAM:
  - Crie um perfil que seu usuário possa assumir. Siga as instruções em Criação de um perfil para um usuário do IAM no Guia do usuário do IAM.
  - (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Forneça AWS credenciais ao IDT

Para permitir que o IDT acesse suas AWS credenciais e envie métricas para AWS, faça o seguinte:

1. Armazene as AWS credenciais do seu usuário do IAM como variáveis de ambiente ou em um arquivo de credenciais:

   a. Para usar as variáveis de ambiente, execute o seguinte comando:

   ```
   AWS_ACCESS_KEY_ID=access-key
   ```

```
AWS_SECRET_ACCESS_KEY=secret-access-key
```

b. Para o arquivo de credenciais, adicione as seguintes informações para `.aws/credentials file:`

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. Configure a seção `auth` do arquivo `config.json`. Para ter mais informações, consulte (Opcional) Configurar config.json.

Versões do pacote de testes do AWS IoT Device Tester para FreeRTOS

O IDT para FreeRTOS organiza testes em pacotes de testes e grupos de testes:

- Um pacote de testes é o conjunto de grupos de teste usado para verificar se um dispositivo funciona com versões específicas do FreeRTOS.
- Um grupo de teste é o conjunto de testes individuais relacionados a um recurso específico, como sistemas de mensagens BLE e MQTT.

Começando com o IDT v3.0.0, os conjuntos de testes são versionados usando um formato `major.minor.patch` começando com 1.0.0. Quando você faz download do IDT, o pacote inclui a versão mais recente do conjunto de testes.

Quando você inicia o IDT na interface de linha de comando, o IDT verifica se uma versão mais recente do conjunto de testes está disponível. Em caso afirmativo, solicitará que você atualize para a nova versão. Você pode optar por atualizar ou continuar com seus testes atuais.

> **Note**
>
> O IDT é compatível com as três versões mais recentes do conjunto de testes para qualificação. Para obter mais informações, consulte Política de suporte do AWS IoT Device Tester para FreeRTOS.

Você pode fazer download dos conjuntos de testes usando o comando `upgrade-test-suite`. Ou pode usar o parâmetro opcional `-upgrade-test-suite` *`flag`* ao iniciar o IDT em que *`flag`* pode ser 'y' para sempre fazer download da versão mais recente, ou 'n' para usar a versão existente.

Também é possível executar o comando `list-supported-versions` para listar as versões do FreeRTOS e do pacote de testes compatíveis com a versão atual do IDT.

Novos testes podem apresentar novas definições de configuração do IDT. Se as configurações forem opcionais, o IDT notificará você e continuará executando os testes. Se as configurações forem necessárias, o IDT notificará você e interromperá a execução. Depois de definir as configurações, você pode continuar a executar os testes.

Solução de problemas

Cada execução do conjunto de testes tem um ID de execução exclusivo que é usado para criar uma pasta chamada `results/`*`execution-id`* no diretório `results`. Os logs individuais do grupo de testes estão no diretório `results/`*`execution-id`*`/logs`. Use a saída do console do IDT para FreeRTOS para encontrar o ID de execução, o ID do caso de teste e o ID do grupo de testes do caso de teste que falhou, e abra o arquivo de log para esse caso de teste chamado `results/`*`execution-id`*`/logs/`*`test_group_id__test_case_id`*`.log`. As informações desse arquivo incluem:

- Saída completa de comando de compilação e flash.
- Saída de execução de teste.
- Mais detalhes da saída de console do IDT para FreeRTOS.

Recomendamos o seguinte fluxo de trabalho para solucionar problemas:

1. Se vir o erro "*`usuário/função`* você não está autorizado a acessar este recurso", certifique-se de configurar permissões conforme especificado em Crie e configure uma AWS conta.
2. Leia a saída do console para encontrar informações, como UUID de execução e tarefas atualmente em execução.
3. Examine o arquivo `FRQ_Report.xml` para verificar se há declarações de erro em cada teste. Esse diretório contém os logs de execução de cada grupo de teste.
4. Procure os arquivos de logs em `/results/`*`execution-id`*`/logs`.
5. Investigue uma das seguintes áreas problemáticas:

- Configuração de dispositivo, como arquivos de configuração JSON na pasta `/configs/`.
- Interface do dispositivo. Verifique os logs para determinar qual interface está falhando.
- Ferramentas do dispositivo. Certifique-se de que os conjuntos de ferramentas para a criação e a atualização do dispositivo estejam instaladas e configuradas corretamente.
- Para o FRQ 1.x.x, certifique-se de que uma versão limpa e clonada do código-fonte do FreeRTOS esteja disponível. Os lançamentos do FreeRTOS são marcados de acordo com a versão do FreeRTOS. Para clonar uma versão específica do código, use os comandos a seguir:

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

Solução de problemas de configuração de dispositivos

Ao usar o IDT para FreeRTOS, é preciso obter os arquivos de configuração corretos antes de executar o binário. Se você estiver recebendo erros de análise e configuração, o primeiro passo deve ser localizar e usar um modelo de configuração apropriado para seu ambiente. Esses modelos estão localizados no diretório *IDT_ROOT*`/configs`.

Se você ainda estiver com problemas, consulte o processo de depuração a seguir.

Onde eu procuro?

Comece lendo a saída do console para encontrar informações, como o UUID de execução, que é referido como `execution-id` nesta documentação.

Depois, examine o arquivo `FRQ_Report.xml` no diretório `/results/`*execution-id*. Este arquivo contém todos os casos de teste que foram executados e os trechos com erros para cada falha. Para obter todos os logs de execução, procure o arquivo `/results/`*execution-id*`/logs/`*test_group_id*`__`*test_case_id*`.log` para cada caso de teste.

Códigos de erro de IDT

A tabela a seguir explica os códigos de erro gerados pelo IDT para FreeRTOS:

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
201	InvalidInputError	Os campos em `device.json` , `config.json` ou `userdata.json` estão ausentes ou em um formato incorreto.	Certifique-se de que os campos obrigatórios não estejam ausentes e estejam no formato obrigatório nos arquivos listados. Para obter mais informações, consulte Preparação para testar sua placa de microcontrolador pela primeira vez.
202	ValidationError	Os campos em `device.json` , `config.json` ou `userdata.json` contêm valores inválidos.	Verifique a mensagem de erro no lado direito do código de erro no relatório: • Região da AWS inválida: especifique uma região da AWS válida em seu arquivo `config.json` . Para obter mais informações sobre as regiões da AWS, consulte Regiões e Endpoints. • Credenciais da AWS inválidas: defina credenciais da AWS válidas em seu computador de

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			teste (com variáveis de ambiente ou arquivo de credenciais). Verifique se o campo de autenticação está configurado corretamente. Para obter mais informações, consulte Crie e configure uma AWS conta.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
203	CopySourceCodeError	Não foi possível copiar o código-fonte do FreeRTOS para o diretório especificado.	Verifique os seguintes itens: • Verifique se um `sourcePath` válido está especificado em seu arquivo `userdata.json` . • Exclua a pasta `build` no diretório do código-fonte do FreeRTOS, se ela existir. Para obter mais informações, consulte Configuração de parâmetros de compilação, atualização e teste. • O Windows tem um limite de caracteres para nomes de caminhos de arquivos. Um nome de caminho de arquivo longo causará um erro.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
204	BuildSourceError	Não foi possível compilar o código-fonte do FreeRTOS.	Verifique os seguintes itens: • Verifique se as informações em `buildTool` no arquivo `userdata.json` estão corretas. • Se você estiver usando o `cmake` como uma ferramenta de compilação, certifique-se de que o `{{enableTests}}` esteja especificado no comando `buildTool` . Para obter mais informações, consulte Configuração de parâmetros de compilação, atualização e teste. • Se extraiu o IDT para FreeRTOS em um caminho de arquivo em seu sistema que contém espaços, por exemplo `C:`

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			`\Users\My Name\Desktop\`, talvez sejam necessárias aspas adicionais dentro de seus comandos de compilação para garantir que os caminhos sejam analisados corretamente. A mesma coisa pode ser necessária em seus comandos flash.
205	FlashOrRunTestError	O IDT FreeRTOS não consegue instalar ou executar o FreeRTOS no seu DUT.	Verifique se as informações em `flashTool` no arquivo `userdata.json` estão corretas. Para obter mais informações, consulte Configuração de parâmetros de compilação, atualização e teste.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
206	StartEchoServerError	O IDT FreeRTOS não consegue iniciar o servidor echo para os testes de Wi-Fi nem de soquetes seguros.	Verifique se as portas configuradas `echoServerConfiguration` em seu arquivo `userdata.json` não estão em uso ou bloqueadas por configurações de firewall ou rede.

Depuração de erros de análise do arquivo de configuração

Ocasionalmente, um erro ortográfico em uma configuração JSON pode resultar em erros de análise. Na maioria dos casos, o problema é resultado da omissão de um colchete, vírgula ou aspas de seu arquivo JSON. O IDT para FreeRTOS executa a validação do JSON e imprime as informações de depuração. Ele imprime a linha em que ocorreu o erro, o número da linha e o número da coluna do erro de sintaxe. Essas informações devem ser suficientes para ajudá-lo a corrigir o erro, mas se você ainda tiver problemas para localizar o erro, poderá executar a validação manualmente no IDE, em um editor de texto como o Atom ou o Sublime, ou por meio de uma ferramenta online, como a JSONLint.

Depuração de erros de análise de resultados de testes

Ao executar um grupo de testes de FreeRTOS-Libraries-Integration-Tests, como FullTransportInterfaceTLS, FullPKCS11_Core, FullPKCS11_Onboard_ECC, FullPKCS11_Onboard_RSA, FullPKCS11_PreProvisioned_ECC, FullPKCS11_PreProvisioned_RSA ou OTACore, o IDT para FreeRTOS analisa os resultados do teste do dispositivo de teste com a conexão serial. Às vezes, saídas seriais extras no dispositivo podem interferir na análise dos resultados do teste.

No caso mencionado acima, motivos estranhos de falha em casos de teste, como cadeias de caracteres provenientes de saídas de dispositivos não relacionados, são gerados. O arquivo de log

do caso de teste do IDT para FreeRTOS (que inclui toda a saída serial que o IDT do FreeRTOS recebeu durante o teste) pode mostrar o seguinte:

```
<unrelated device output>
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities)<unrelated device output>
<unrelated device output>
 PASS
```

No exemplo acima, a saída não relacionada do dispositivo impede que o IDT para FreeRTOS detecte o resultado do teste, que é APROVADO.

Verifique o seguinte para garantir o teste ideal.

- Verifique se as macros de registro em log usadas no dispositivo são seguras para thread. Consulte Implementação das macros de registro em log da biblioteca para obter mais informações.
- Verifique se há um mínimo de saídas para a conexão serial durante os testes. As saídas de outros dispositivos podem ser um problema, mesmo que suas macros de registro em log sejam devidamente seguras, pois os resultados do teste serão exibidos em chamadas separadas durante o teste.

Idealmente, um log de casos de teste do IDT para FreeRTOS mostraria uma saída ininterrupta dos resultados do teste, como abaixo:

```
---------STARTING TESTS---------
TEST(Full_OTA_PAL, otaPal_CloseFile_ValidSignature) PASS
TEST(Full_OTA_PAL, otaPal_CloseFile_InvalidSignatureBlockWritten) PASS
-----------------------
2 Tests 0 Failures 0 Ignored
```

Como depurar as falhas na verificação de integridade

Se estiver usando a versão FRQ 1.x.x dos FreeRTOS, as verificações de integridade a seguir se aplicam.

Ao executar o grupo de teste FreeRtosIntegrity e encontrar falhas, verifique primeiro se você não modificou nenhum dos arquivos do diretório *`freertos`*. Se não fez isso e ainda está tendo

problemas, verifique se está usando a ramificação correta. Se executar o comando `list-supported-products` do IDT, poderá descobrir qual ramificação marcada do repositório *`freertos`* deve ser usada.

Se clonou a ramificação marcada correta do repositório `freertos` e ainda tiver problemas, verifique se também executou o comando `submodule update`. O fluxo de trabalho de clonagem para o repositório `freertos` é o seguinte.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout —init —recursive
```

A lista de arquivos que o verificador de integridade procura está no arquivo `checksums.json` em seu diretório *`freertos`*. Para qualificar uma porta do FreeRTOS sem nenhuma modificação nos arquivos e na estrutura de pastas, verifique se nenhum dos arquivos listados nas seções "`exhaustive`" e "`minimal`" do arquivo `checksums.json` foi modificado. Para executar com um SDK configurado, verifique se nenhum dos arquivos na seção "`minimal`" foi modificado.

Se o IDT for executado com um SDK e alguns arquivos em seu diretório *`freertos`* tiverem sido modificados, certifique-se de configurar corretamente o SDK em seu arquivo `userdata`. Caso contrário, o verificador de integridade verificará todos os arquivos no diretório *`freertos`*.

Como depurar falhas do grupo de testes FullWifi

Se estiver usando o FRQ 1.x.x e encontrar falhas no grupo de teste FullWiFi e o teste "`AFQP_WiFiConnectMultipleAP`" falhar, pode ser porque os dois pontos de acesso não estão na mesma sub-rede que o computador host executando o IDT. Verifique se os dois pontos de acesso estão na mesma sub-rede do computador host que executa o IDT.

Depuração de erro de "parâmetro necessário ausente"

Como novos atributos estão sendo adicionados ao IDT para FreeRTOS, os arquivos de configuração podem sofrer alterações. O uso de um arquivo de configuração antigo pode danificar sua configuração. Se isso acontecer, o arquivo *`test_group_id__test_case_id`*`.log` no diretório `results/`*`execution-id`*`/logs` listará explicitamente todos os parâmetros ausentes. O IDT para FreeRTOS valida os esquemas do arquivo de configuração JSON para garantir que a versão compatível mais recente foi usada.

Como depurar um erro de "não foi possível iniciar teste"

Você pode encontrar erros que apontam para falhas ao iniciar o teste. Como existem várias causas possíveis, verifique se há algum problema nas seguintes áreas:

- Verifique se o nome do grupo incluído no comando de execução realmente existe. Ele é referenciado diretamente em seu arquivo `device.json`.
- Verifique se o(s) dispositivo(s) no grupo têm os parâmetros de configuração corretos.

Como depurar um erro "não foi possível encontrar o início dos resultados do teste"

Você pode obter erros quando o IDT tenta analisar a saída de resultados pelo dispositivo em teste. Existem várias causas possíveis, por isso verifique se há algum problema nas seguintes áreas:

- Verifique se o dispositivo em teste tem uma conexão estável com sua máquina host. É possível verificar o arquivo de log para ver se há um teste que mostre esses erros e ver o que o IDT está recebendo.
- Se estiver usando o FRQ 1.x.x e o dispositivo em teste estiver conectado por meio de uma rede lenta ou outra interface, ou se você não ver o sinalizador "---------STARTING TESTS---------" em um log do grupo de testes do FreeRTOS junto com outras saídas do grupo de testes do FreeRTOS, tente aumentar o valor de `testStartDelayms` em sua configuração de dados do usuário. Para obter mais informações, consulte Configuração de parâmetros de compilação, atualização e teste.

Como depurar um erro "Falha no teste: resultados esperados __, mas foram vistos ___"

Você pode encontrar erros que apontam para falhas ao iniciar o teste. O teste espera um certo número de resultados e não vê isso durante o teste. Alguns testes do FreeRTOS são executados antes que o IDT veja a saída do dispositivo. Se ver esse erro, tente aumentar o valor de `testStartDelayms` em sua configuração de dados de usuário. Para obter mais informações, consulte Configuração de parâmetros de compilação, atualização e teste.

Como depurar um erro "_____ não foi selecionado devido às restrições do ConditionalTests"

Isso significa que você está executando um teste em um grupo de dispositivos que é incompatível com o teste. Isso pode acontecer com os testes E2E OTA. Por exemplo, ao executar o grupo de

testes `OTADataplaneMQTT` e em seu arquivo de configuração `device.json`, você escolheu OTA como Não ou `OTADataPlaneProtocol` como HTTP. O grupo de teste escolhido para execução deve corresponder às suas seleções de capacidade `device.json`.

Como depurar um tempo limite do IDT durante o monitoramento de saída do dispositivo

O IDT pode atingir o tempo limite por conta de vários motivos. Se ocorrer um tempo limite durante a fase de monitoramento da saída do dispositivo de um teste e puder ver os resultados no log do caso de testes do IDT, isso significa que os resultados foram analisados incorretamente pelo IDT. Um dos motivos podem ser as mensagens de log intercaladas no meio dos resultados do teste. Se for este o caso, consulte o Guia de portabilidade do FreeRTOS para obter mais detalhes sobre como os logs do UNITY devem ser configurados.

Outro motivo para o tempo limite durante o monitoramento da saída do dispositivo pode ser a reinicialização do dispositivo após uma única falha no caso de testes do TLS. O dispositivo executará a imagem instalada e causará um loop infinito que será visto nos logs. Se isto acontecer, o seu dispositivo não será reinicializado após uma falha no teste.

Depuração de erro de "não autorizado para acessar um recurso"

Você poderá ver o erro "*`usuário/função`* não está autorizado a acessar este recurso" na saída do terminal ou no arquivo `test_manager.log` em `/results/`*`execution-id`*`/logs`. Para resolver este problema, anexe a política gerenciada `AWSIoTDeviceTesterForFreeRTOSFullAccess` ao usuário de teste. Para obter mais informações, consulte Crie e configure uma AWS conta.

Depuração de erros de teste de rede

Para testes com base em rede, o IDT inicia um servidor de eco que vincula a uma porta não reservada na máquina host. Se você estiver se deparando com erros devido a tempos limite ou conexões indisponíveis nos testes de Wi-Fi ou de soquetes seguros, verifique se a rede está configurada para permitir tráfego nas portas configuradas no intervalo 1024–49151.

O teste de soquetes seguros usa as portas 33333 e 33334 por padrão. Os testes de Wi-Fi usam a porta 33335 por padrão. Se essas três portas estiverem em uso ou bloqueadas por firewall ou rede, opte por usar portas diferentes em userdata.json para testes. Para obter mais informações, consulte Configuração de parâmetros de compilação, atualização e teste. Você pode usar os seguintes comandos para verificar se uma porta específica está em uso:

- Windows: `netsh advfirewall firewall show rule name=all | grep port`

- Linux: `sudo netstat -pan | grep port`
- macOS: `netstat -nat | grep port`

Falhas de atualização de OTA devido à mesma carga útil de versão

Se os casos de teste OTA estiverem falhando por a mesma versão estar no dispositivo depois que uma OTA foi executada, pode ser que o seu sistema de compilação (por exemplo, cmake) não notou as alterações do IDT ao código-fonte do FreeRTOS e não criou um binário atualizado. Isso faz com que o OTA seja executado com o mesmo binário que está atualmente no dispositivo e o teste falhe. Para solucionar problemas de falhas de atualização OTA, comece certificando-se de que você está usando a versão mais recente com suporte do sistema de compilação.

Falha no teste de OTA no caso de teste de **`PresignedUrlExpired`**

Um pré-requisito deste teste é que o tempo de atualização OTA deve ser superior a 60 segundos, caso contrário o teste falharia. Se isso ocorrer, a seguinte mensagem de erro é encontrada no log: “Teste leva menos de 60 segundos (tempo expirado da url) para concluir. Por favor, entre em contato conosco."

Depuração erros de porta e de interface de dispositivo

Esta seção contém informações sobre as interfaces de dispositivo usadas pelo IDT para se conectar aos dispositivos.

Plataformas compatíveis

O IDT é compatível com Linux, macOS e Windows. Todas as três plataformas têm diferentes esquemas de nomenclatura para os dispositivos seriais que se conectam a elas:

- Linux: `/dev/tty*`
- macOS: `/dev/tty.*` ou `/dev/cu.*`
- Windows: COM*

Para verificar a porta do dispositivo:

- No Linux/macOS, abra um terminal e execute `ls /dev/tty*`.
- No macOS, abra um terminal e execute `ls /dev/tty.*` ou `ls /dev/cu.*`.
- No Windows, abra o Gerenciador de dispositivos e expanda o grupo de dispositivos seriais.

Para verificar qual dispositivo está conectado a uma porta:

- Para o Linux, verifique se o pacote `udev` está instalado e execute `udevadm info –name=`*`PORT`*. Este utilitário imprime informações do driver de dispositivo que ajudam você a verificar se está usando a porta correta.
- Para macOS, abra o Launchpad e procure **`System Information`**.
- No Windows, abra o Gerenciador de dispositivos e expanda o grupo de dispositivos seriais.

Interfaces de dispositivo

Cada dispositivo incorporado é diferente, o que significa que ele pode ter uma ou mais portas seriais. É comum que os dispositivos tenham duas portas quando conectados a uma máquina:

- Uma porta de dados para a atualização do dispositivo.
- Uma porta de leitura para ler a saída.

  Você deve definir a porta de leitura correta em seu arquivo `device.json`. Caso contrário, a leitura de saída do dispositivo pode falhar.

  No caso de várias portas, certifique-se de usar a porta de leitura do dispositivo em seu arquivo `device.json`. Por exemplo, se você conectar um dispositivo Espressif WRover e as duas portas atribuídas a ele forem `/dev/ttyUSB0` e `/dev/ttyUSB1`, use `/dev/ttyUSB1` em seu arquivo `device.json`.

  No Windows, siga a mesma lógica.

Leitura de dados de dispositivo

O IDT para FreeRTOS usa ferramentas individuais de compilação e atualização de dispositivos para especificar a configuração da porta. Se você estiver testando o dispositivo e não obtiver a saída, tente as seguintes configurações padrão:

- Taxa de baud: 115200
- Bits de dados: 8
- Paridade: nenhum
- Bits de parada: 1
- Controle de fluxo: nenhum

Essas configurações são processadas pelo IDT para FreeRTOS. Você não precisa defini-los. No entanto, você pode usar o mesmo método para ler a saída do dispositivo manualmente. No Linux ou macOS, você pode fazer isso com o comando `screen`. No Windows, você pode usar um programa como o TeraTerm.

```
Screen: screen /dev/cu.usbserial 115200
```

```
TeraTerm: Use the above-provided settings to set the fields explicitly in
the GUI.
```

Problemas de cadeia de ferramentas de desenvolvimento

Esta seção aborda os problemas que podem ocorrer com a cadeia de ferramentas.

Code Composer Studio no Ubuntu

As versões mais recentes do Ubuntu (17.10 e 18.04) têm uma versão do pacote `glibc` que não é compatível com as versões 7.x do Code Composer Studio. Recomendamos que você instale o Code Composer Studio versão 8.2 ou mais recente.

Os sintomas de incompatibilidade podem incluir:

- Falha do FreeRTOS ao compilar ou atualizar seu dispositivo.
- O instalador do Code Composer Studio pode congelar.
- Nenhuma saída de log é exibida no console durante o processo de compilação ou atualização.
- O comando de compilação tenta executar em modo GUI mesmo quando é invocado em modo dedicado.

Registro em log

Os logs do IDT para FreeRTOS são armazenados em um único local. No diretório raiz do IDT, esses arquivos estão disponíveis em `results/`*`execution-id`*`/`:

- `FRQ_Report.xml`
- `awsiotdevicetester_report.xml`
- `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`

`FRQ_Report.xml` e `logs/`*`test_group_id`*`__`*`test_case_id`*`.log` são os logs mais importantes a serem examinados. `FRQ_Report.xml` contém informações sobre quais

casos de teste falharam com uma mensagem de erro específica. Depois, é possível usar `logs/test_group_id__test_case_id.log` para ver mais detalhes sobre o problema, a fim de obter um contexto melhor.

Erros do console

Quando o AWS IoT Device Tester é executado, as falhas são exibidas no console com mensagens breves. Examine `results/execution-id/logs/test_group_id__test_case_id.log` para saber mais sobre o erro.

Erros de logs

Cada execução do conjunto de testes tem um ID de execução exclusivo usado para criar uma pasta chamada `results/execution-id`. Os logs de casos de testes individuais estão no diretório `results/execution-id/logs`. Use a saída do console do IDT para FreeRTOS a fim de localizar o ID de execução, o ID de caso de teste e o ID de grupo de teste do caso de teste que falhou. Depois, use essas informações para localizar e abrir o arquivo de log para este caso de testes chamado `results/execution-id/logs/test_group_id__test_case_id.log`. As informações nesse arquivo incluem a saída completa do comando compilar e atualizar, a saída da execução do teste e a saída mais detalhada do console do AWS IoT Device Tester Device Tester.

Problemas de bucket do S3

Se pressionar CTRL+C enquanto executa o IDT, o IDT iniciará um processo de limpeza. Parte dessa limpeza é remover os recursos do Amazon S3 que foram criados como parte dos testes do IDT. Se a limpeza não puder ser concluída, você poderá ter um problema em que muitos buckets do Amazon S3 foram criados. Isso significa que na próxima vez que você executar o IDT, os testes começarão a falhar.

Se pressionar CTRL+C para interromper o IDT, deverá deixá-lo concluir o processo de limpeza para evitar este problema. Você também pode excluir os buckets do Amazon S3 da sua conta que foram criados manualmente.

Solução de erros de tempo limite

Se você encontrar erros de tempo limite ao executar um conjunto de testes, aumente o tempo limite especificando um fator multiplicador. Esse fator é aplicado ao valor de tempo limite padrão. Qualquer valor configurado para esse sinalizador deve ser maior que ou igual a 1.0. Para usar o multiplicador de tempo limite, use o sinalizador `--timeout-multiplier` ao executar o conjunto de testes.

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
```

Atributo de celular e cobranças da AWS

Quando o atributo `Cellular` estiver definido como `Yes` no seu arquivo `device.JSON`, o FullSecureSockets será executado usando instâncias t.micro do EC2 para executar testes e isso poderá gerar custos adicionais para sua conta da AWS. Para obter mais informações, consulte Definição de preço do Amazon EC2.

Política de geração de relatórios de qualificação

Os relatórios de qualificação são gerados somente pelas versões do AWS IoT Device Tester (IDT) que oferecem suporte às versões do FreeRTOS lançadas nos últimos dois anos. Se tiver dúvidas sobre a política de suporte, entre em contato com o AWS Support.

Política gerenciada da AWS para o AWS IoT Device Tester

Uma política gerenciada pela AWS é uma política independente criada e administrada pela AWS. As políticas gerenciadas pela AWS são criadas para fornecer permissões a vários casos de uso comuns a fim de que você possa começar a atribuir permissões a usuários, grupos e perfis.

Lembre-se de que as políticas gerenciadas pela AWS podem não conceder permissões de privilégio mínimo para seus casos de uso específicos porque estão disponíveis para todos os clientes da AWS usarem. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo cliente específicas para seus casos de uso.

Você não pode alterar as permissões definidas em políticas gerenciadas pela AWS. Se a AWS atualiza as permissões definidas em um política gerenciada pela AWS, a atualização afeta todas as identidades de entidades principais (usuários, grupos e perfis) às quais a política está

vinculada. É mais provável que a AWS atualize uma política gerenciada pela AWS quando um novo Serviço da AWS é lançado ou novas operações de API são disponibilizadas para os serviços existentes.

Para obter mais informações, consulte Políticas gerenciadas pela AWS no Manual do usuário do IAM.

Tópicos

Política gerenciada da AWS: IoTDeviceTesterForFreeRTOSFullAccess da AWS

A política gerenciada `AWSIoTDeviceTesterForFreeRTOSFullAccess` contém as seguintes permissões do AWS IoT Device Tester para a verificação de versão, atributos de atualização automática e coleção de métricas.

Detalhes da permissão

Esta política inclui as seguintes permissões:

- `iot-device-tester:SupportedVersion`

  Concede permissão ao AWS IoT Device Tester para buscar a lista de produtos compatíveis, pacotes de teste e versões do IDT.
- `iot-device-tester:LatestIdt`

  Concede permissão ao AWS IoT Device Tester para obter a versão mais recente do IDT disponível para download.
- `iot-device-tester:CheckVersion`

  Concede permissão ao AWS IoT Device Tester para verificar a compatibilidade de versões do IDT, pacotes de teste e produtos.
- `iot-device-tester:DownloadTestSuite`

  Concede permissão ao AWS IoT Device Tester para fazer download de pacotes de teste.
- `iot-device-tester:SendMetrics`

Concede permissão à AWS para coletar métricas sobre o uso interno do 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"
```

```
                    ]
                }
            }
        }
    ]
}
```

Atualizações do AWS IoT Device Tester para políticas gerenciadas pela AWS

É possível visualizar detalhes sobre atualizações em políticas gerenciadas daAWS para o AWS IoT Device Tester do momento que esse serviço começou a rastrear essas alterações.

Versão	Alteração	Descrição	Data
7 (Mais recente)	Reestruturou as condições `ec2:CreateTags` .	Remoção do uso de `ForAnyValues` .	14/06/2023
6	Removeu `freertos:ListHardwarePlatforms` da política.	Remoção das permissões, pois essa ação foi preterida em 1º de março de 2023.	6/02/2023
5	Adicionou permissões para executar testes de servidor echo usando o EC2.	Isso serve para iniciar e interromper uma instância do EC2 nas contas dos clientes da AWS.	15/12/2020
4	Adição do `iot:CancelJobExecution` .	Essa permissão cancela trabalhos OTA.	17/07/2020
3	Adicionou as seguintes permissões: • `iot-device-tester:`	• `iot-device-tester:DownloadTestSuite` : concede permissão	23/03/2020

Versão	Alteração	Descrição	Data
	`DownloadTestSuite`, • `iot-device-tester:CheckVersion`, • `iot-device-tester:LatestIdt`, • `iot-device-tester:SupportedVersion`.	ao AWS IoT Device Tester para fazer download de pacotes de teste, • `iot-device-tester:CheckVersion`: concede permissão ao AWS IoT Device Tester para verificar a compatibilidade para o IDT, pacotes de teste e produtos, • `iot-device-tester:LatestIdt`: concede permissão ao AWS IoT Device Tester para obter a versão mais recente do IDT disponível para download, • `iot-device-tester:SupportedVersion`: concede permissão ao AWS IoT Device Tester para buscar a lista de produtos compatíveis, pacotes de teste e versões do IDT.	

Versão	Alteração	Descrição	Data
2	Adicionou permissões de `iot-device-tester:SendMetrics` .	Concede permissão à AWS para coletar métricas sobre o uso interno do AWS IoT Device Tester.	18/02/2020
1	Versão inicial.		12/02/2020

Política de suporte do AWS IoT Device Tester para FreeRTOS

> **Important**
>
> Desde outubro de 2022, o AWS IoT Device Tester para AWS IoT da qualificação do FreeRTOS (FRQ) 1.0 não gera relatórios de qualificação assinados. Você não pode qualificar novos dispositivos AWS IoT do FreeRTOS para serem listados no AWS Partner Device Catalog por meio do Programa de qualificação de dispositivos da AWS usando as versões IDT FRQ 1.0. Embora você não possa qualificar dispositivos FreeRTOS usando o IDT FRQ 1.0, você pode continuar testando seus dispositivos FreeRTOS com o FRQ 1.0. Recomendamos que você use o IDT FRQ 2.0 para qualificar e listar dispositivos FreeRTOS no AWS Partner Device Catalog.

O AWS IoT Device Tester para FreeRTOS é uma ferramenta de automação de testes para validar a porta do FreeRTOS para dispositivos. Além disso, você pode qualificar seus dispositivos FreeRTOS e listá-los no AWS Partner Device Catalog. O AWS IoT Device Tester para FreeRTOS oferece suporte à validação e qualificação das bibliotecas de suporte de longo prazo (LTS) do FreeRTOS disponíveis no GitHub em FreeRTOS/FreeRTOS-LTS e na linha principal do FreeRTOS disponível em FreeRTOS/FreeRTOS. Recomendamos que você use as versões mais recentes do FreeRTOS e do AWS IoT Device Tester para FreeRTOS para validar e qualificar seus dispositivos.

Para FreeRTOS-LTS, o IDT oferece suporte a validação e qualificação da versão 202210 LTS do FreeRTOS. Veja aqui mais informações sobre as Versões LTS do FreeRTOS e seu cronograma de manutenção. Quando o período de suporte dessas versões do LTS termina, você ainda poderá continuar a validação, mas o IDT não gerará um relatório que permita enviar seu dispositivo para qualificação.

Para os linhas principais do FreeRTOS disponíveis no FreeRTOS/FreeRTOS, oferecemos suporte à validação e qualificação de todas as versões lançadas nos últimos seis meses, ou das duas versões anteriores do FreeRTOS, se lançadas com mais de seis meses de intervalo. Confira aqui as versões compatíveis no momento. Para as versões do FreeRTOS sem suporte, você ainda poderá continuar a validação, mas o IDT não gerará um relatório que permita enviar seu dispositivo para qualificação.

Consulte Versões compatíveis do AWS IoT Device Tester para FreeRTOS as versões compatíveis do IDT e do FreeRTOS mais recentes. É possível usar qualquer uma das versões compatíveis do AWS IoT Device Tester com a versão correspondente do FreeRTOS para testar ou qualificar seus dispositivos. Se continuar a usar Versões não compatíveis do IDT para FreeRTOS, você não receberá as correções de bugs ou atualizações mais recentes.

Para tirar dúvidas sobre a política de suporte, entre em contato com o Suporte ao cliente da AWS.

Segurança em AWS

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de uma arquitetura de data center e rede criada para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isto como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços na AWS nuvem. AWS também fornece serviços que você pode usar com segurança. A eficácia da nossa segurança é regularmente testada e verificada por auditores de terceiros como parte dos Programas de conformidade da AWS. Para saber mais sobre os programas de conformidade que se aplicam a um AWS serviço, consulte AWS Serviços no escopo por programa de conformidade.
- Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você usa. Você também é responsável por outros fatores, como a confidencialidade de seus dados, os requisitos da sua organização, leis e regulamentos aplicáveis.

Esta documentação ajudará você a entender como aplicar o modelo de responsabilidade compartilhada ao usar AWS. Os tópicos a seguir mostram como configurar para atender AWS aos seus objetivos de segurança e conformidade. Você também aprenderá a usar AWS serviços que podem ajudá-lo a monitorar e proteger seus AWS recursos.

Para obter informações mais detalhadas sobre AWS IoT segurança, consulte Segurança e identidade para AWS IoT.

Tópicos

Identity and Access Management gratuitamente RTOS

AWS Identity and Access Management (IAM) é uma ferramenta Serviço da AWS que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. IAMos administradores controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) a usar os recursos gratuitosRTOS. IAMé um Serviço da AWS que você pode usar sem custo adicional.

Tópicos

Público

A forma como você usa AWS Identity and Access Management (IAM) difere, dependendo do trabalho que você faz no FreeRTOS.

Usuário do serviço — Se você usar o RTOS serviço gratuito para fazer seu trabalho, seu administrador fornecerá as credenciais e as permissões de que você precisa. À medida que você usa mais RTOS recursos gratuitos para fazer seu trabalho, talvez precise de permissões adicionais. Entender como o acesso é gerenciado pode ajudar você a solicitar as permissões corretas ao seu administrador. Se você não conseguir acessar um recurso no FreeRTOS, consulteSolução de problemas de RTOS identidade e acesso gratuitos.

Administrador de serviços — Se você é responsável pelos RTOS recursos gratuitos em sua empresa, provavelmente tem acesso total aos recursos gratuitosRTOS. É seu trabalho determinar quais RTOS recursos e recursos gratuitos seus usuários do serviço devem acessar. Em seguida, você deve enviar solicitações ao IAM administrador para alterar as permissões dos usuários do serviço. Revise as informações nesta página para entender os conceitos básicos doIAM. Para saber mais sobre como sua empresa pode usar IAM o FreeRTOS, consulteComo o Free RTOS funciona com IAM.

IAMadministrador — Se você for IAM administrador, talvez queira saber detalhes sobre como criar políticas para gerenciar o acesso ao FreeRTOS. Para ver exemplos de políticas gratuitas RTOS

baseadas em identidade que você pode usar emIAM, consulte. Exemplos gratuitos de políticas baseadas em identidade RTOS

Autenticando com identidades

A autenticação é como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado (conectado AWS) como IAM usuário ou assumindo uma IAM função. Usuário raiz da conta da AWS

Você pode entrar AWS como uma identidade federada usando credenciais fornecidas por meio de uma fonte de identidade. AWS IAM Identity Center Os usuários (do IAM Identity Center), a autenticação de login único da sua empresa e suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você entra como uma identidade federada, seu administrador configurou previamente a federação de identidades usando IAM funções. Ao acessar AWS usando a federação, você está assumindo indiretamente uma função.

Dependendo do tipo de usuário que você é, você pode entrar no AWS Management Console ou no portal de AWS acesso. Para obter mais informações sobre como fazer login em AWS, consulte Como fazer login Conta da AWS no Guia do Início de Sessão da AWS usuário.

Se você acessar AWS programaticamente, AWS fornece um kit de desenvolvimento de software (SDK) e uma interface de linha de comando (CLI) para assinar criptograficamente suas solicitações usando suas credenciais. Se você não usa AWS ferramentas, você mesmo deve assinar as solicitações. Para obter mais informações sobre como usar o método recomendado para você mesmo assinar solicitações, consulte Assinar AWS API solicitações no Guia IAM do usuário.

Independente do método de autenticação usado, também pode ser exigido que você forneça informações adicionais de segurança. Por exemplo, AWS recomenda que você use a autenticação multifator (MFA) para aumentar a segurança da sua conta. Para saber mais, consulte Autenticação multifator no Guia AWS IAM Identity Center do usuário e Uso da autenticação multifator (MFA) AWS no Guia do IAMusuário.

Conta da AWS usuário root

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos Serviços da AWS os recursos da conta. Essa identidade é chamada de usuário Conta da AWS raiz e é acessada fazendo login com o endereço de e-mail e a senha que você usou para criar a conta. É altamente recomendável não usar o usuário raiz para tarefas diárias. Proteja as credenciais do usuário raiz e use-as para executar as tarefas que somente ele puder executar. Para ver a lista

completa de tarefas que exigem que você faça login como usuário raiz, consulte Tarefas que exigem credenciais de usuário raiz no Guia do IAM usuário.

Identidade federada

Como prática recomendada, exija que usuários humanos, incluindo usuários que precisam de acesso de administrador, usem a federação com um provedor de identidade para acessar Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório de usuários corporativo, de um provedor de identidade da web AWS Directory Service, do diretório do Identity Center ou de qualquer usuário que acesse usando credenciais fornecidas Serviços da AWS por meio de uma fonte de identidade. Quando as identidades federadas são acessadas Contas da AWS, elas assumem funções, e as funções fornecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos usar o AWS IAM Identity Center. Você pode criar usuários e grupos no IAM Identity Center ou pode se conectar e sincronizar com um conjunto de usuários e grupos em sua própria fonte de identidade para uso em todos os seus Contas da AWS aplicativos. Para obter informações sobre o IAM Identity Center, consulte O que é o IAM Identity Center? no Guia do AWS IAM Identity Center usuário.

Grupos e usuários do IAM

Um IAMusuário é uma identidade dentro da sua Conta da AWS que tem permissões específicas para uma única pessoa ou aplicativo. Sempre que possível, recomendamos confiar em credenciais temporárias em vez de criar IAM usuários que tenham credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de uso específicos que exijam credenciais de longo prazo com IAM os usuários, recomendamos que você alterne as chaves de acesso. Para obter mais informações, consulte Alterne as chaves de acesso regularmente para casos de uso que exigem credenciais de longo prazo no Guia do IAMusuário.

Um IAMgrupo é uma identidade que especifica uma coleção de IAM usuários. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdminse conceder a esse grupo permissões para administrar IAM recursos.

Usuários são diferentes de perfis. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários

têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Quando criar um IAM usuário (em vez de uma função) no Guia do IAM usuário.

IAMfunções

Uma IAMfunção é uma identidade dentro da sua Conta da AWS que tem permissões específicas. É semelhante a um IAM usuário, mas não está associado a uma pessoa específica. Você pode assumir temporariamente uma IAM função no AWS Management Console trocando de funções. Você pode assumir uma função chamando uma AWS API operação AWS CLI or ou usando uma personalizadaURL. Para obter mais informações sobre métodos de uso de funções, consulte Usando IAM funções no Guia IAM do usuário.

IAMfunções com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, você pode criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas pelo mesmo. Para obter informações sobre funções para federação, consulte Criação de uma função para um provedor de identidade terceirizado no Guia IAM do usuário. Se você usa o IAM Identity Center, configura um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o IAM Identity Center correlaciona o conjunto de permissões a uma função em. IAM Para obter informações sobre conjuntos de permissões, consulte Conjuntos de Permissões no Manual do Usuário do AWS IAM Identity Center .
- Permissões temporárias IAM de IAM usuário — Um usuário ou função pode assumir uma IAM função para assumir temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas — Você pode usar uma IAM função para permitir que alguém (um diretor confiável) em uma conta diferente acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, com alguns Serviços da AWS, você pode anexar uma política diretamente a um recurso (em vez de usar uma função como proxy). Para saber a diferença entre funções e políticas baseadas em recursos para acesso entre contas, consulte Acesso a recursos entre contas IAM no Guia do IAM usuário.
- Acesso entre serviços — Alguns Serviços da AWS usam recursos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicativos na Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões do principal de chamada, usando um perfil de serviço ou um perfil vinculado a serviço.

- Sessões de acesso direto (FAS) — Quando você usa um IAM usuário ou uma função para realizar ações em AWS, você é considerado principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. FASusa as permissões do diretor chamando um Serviço da AWS, combinadas com a solicitação Serviço da AWS para fazer solicitações aos serviços posteriores. FASas solicitações são feitas somente quando um serviço recebe uma solicitação que requer interações com outros Serviços da AWS ou com recursos para ser concluída. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer FAS solicitações, consulte Encaminhar sessões de acesso.
- Função de serviço — Uma função de serviço é uma IAMfunção que um serviço assume para realizar ações em seu nome. Um IAM administrador pode criar, modificar e excluir uma função de serviço internamenteIAM. Para obter mais informações, consulte Criação de uma função para delegar permissões a uma Serviço da AWS no Guia do IAM usuário.
- Função vinculada ao serviço — Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. Serviço da AWS O serviço pode presumir a função de executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um IAM administrador pode visualizar, mas não editar, as permissões das funções vinculadas ao serviço.

- Aplicativos em execução na Amazon EC2 — Você pode usar uma IAM função para gerenciar credenciais temporárias para aplicativos que estão sendo executados em uma EC2 instância e fazendo AWS CLI AWS API solicitações. Isso é preferível a armazenar chaves de acesso na EC2 instância. Para atribuir uma AWS função a uma EC2 instância e disponibilizá-la para todos os aplicativos, você cria um perfil de instância anexado à instância. Um perfil de instância contém a função e permite que programas em execução na EC2 instância recebam credenciais temporárias. Para obter mais informações, consulte Como usar uma IAM função para conceder permissões a aplicativos executados em EC2 instâncias da Amazon no Guia IAM do usuário.

Para saber se usar IAM funções ou IAM usuários, consulte Quando criar uma IAM função (em vez de um usuário) no Guia do IAM usuário.

Gerenciando acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política é um objeto AWS que, quando associada a uma identidade ou recurso, define suas permissões. AWS avalia essas políticas quando um principal (usuário, usuário raiz ou sessão de função) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida

ou negada. A maioria das políticas é armazenada AWS como JSON documentos. Para obter mais informações sobre a estrutura e o conteúdo dos documentos de JSON política, consulte Visão geral das JSON políticas no Guia IAM do usuário.

Os administradores podem usar AWS JSON políticas para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e funções não têm permissões. Para conceder permissão aos usuários para realizar ações nos recursos de que precisam, um IAM administrador pode criar IAM políticas. O administrador pode então adicionar as IAM políticas às funções e os usuários podem assumir as funções.

IAMas políticas definem permissões para uma ação, independentemente do método usado para realizar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de função do AWS Management Console AWS CLI, do ou do AWS API.

Políticas baseadas em identidade

Políticas baseadas em identidade são documentos de políticas de JSON permissões que você pode anexar a uma identidade, como um IAM usuário, grupo de usuários ou função. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Criação de IAM políticas no Guia do IAMusuário.

As políticas baseadas em identidade podem ser categorizadas ainda adicionalmente como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou perfil. As políticas gerenciadas são políticas autônomas que você pode associar a vários usuários, grupos e funções em seu Conta da AWS. As políticas AWS gerenciadas incluem políticas gerenciadas e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte Escolha entre políticas gerenciadas e políticas em linha no Guia do IAMusuário.

Políticas baseadas no recurso

Políticas baseadas em recursos são documentos JSON de política que você anexa a um recurso. Exemplos de políticas baseadas em recursos são políticas de confiança de IAM funções e políticas de bucket do Amazon S3. Em serviços que suportem políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o recurso ao qual a política está anexada, a política define quais ações um principal especificado

pode executar nesse recurso e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. Os diretores podem incluir contas, usuários, funções, usuários federados ou. Serviços da AWS

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas de uma política baseada IAM em recursos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLssão semelhantes às políticas baseadas em recursos, embora não usem o formato de documento JSON de política.

Amazon S3, AWS WAF, e Amazon VPC são exemplos de serviços que oferecem suporte. ACLs Para saber maisACLs, consulte a visão geral da lista de controle de acesso (ACL) no Guia do desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- Limites de permissões — Um limite de permissões é um recurso avançado no qual você define as permissões máximas que uma política baseada em identidade pode conceder a uma IAM entidade (IAMusuário ou função). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade com seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou o perfil no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte Limites de permissões para IAM entidades no Guia IAM do usuário.
- Políticas de controle de serviço (SCPs) — SCPs são JSON políticas que especificam as permissões máximas para uma organização ou unidade organizacional (OU) em AWS Organizations. AWS Organizations é um serviço para agrupar e gerenciar centralmente vários Contas da AWS que sua empresa possui. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as suas contas. Os SCP limites de permissões para entidades nas contas dos membros, incluindo cada uma Usuário raiz da conta da AWS. Para obter mais informações sobre Organizations eSCPs, consulte Políticas de controle de serviços no Guia AWS Organizations do Usuário.

- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em atributo. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte Políticas de sessão no Guia IAM do usuário.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de política estão envolvidos, consulte Lógica de avaliação de políticas no Guia IAM do usuário.

Como o Free RTOS funciona com IAM

Antes de usar IAM para gerenciar o acesso ao GratuitoRTOS, saiba quais IAM recursos estão disponíveis para uso com o GratuitoRTOS.

IAMrecursos que você pode usar com o Free RTOS

IAMrecurso	RTOSSuporte gratuito
Políticas baseadas em identidade	Sim
Políticas baseadas em recursos	Não
Ações das políticas	Sim
Atributos de políticas	Sim
Chaves de condição de política (específicas do serviço)	Sim
ACLs	Não
ABAC(tags nas políticas)	Parcial

IAMrecurso	RTOSSuporte gratuito
Credenciais temporárias	Sim
Permissões de entidade principal	Sim
Perfis de serviço	Sim
Perfis vinculados ao serviço	Não

Para ter uma visão geral de como os AWS serviços gratuitos RTOS e outros funcionam com a maioria dos IAM recursos, consulte AWS os serviços que funcionam com IAM no Guia do IAM usuário.

Políticas baseadas em identidade gratuitas RTOS

Compatível com políticas baseadas em identidade: Sim

Políticas baseadas em identidade são documentos de políticas de JSON permissões que você pode anexar a uma identidade, como um IAM usuário, grupo de usuários ou função. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Criação de IAM políticas no Guia do IAMusuário.

Com políticas IAM baseadas em identidade, você pode especificar ações e recursos permitidos ou negados, bem como as condições sob as quais as ações são permitidas ou negadas. Você não pode especificar a entidade principal em uma política baseada em identidade porque ela se aplica ao usuário ou perfil ao qual ela está anexada. Para saber mais sobre todos os elementos que você pode usar em uma JSON política, consulte a referência IAM JSON de elementos de política no Guia IAM do usuário.

Exemplos gratuitos de políticas baseadas em identidade RTOS

Para ver exemplos de políticas gratuitas RTOS baseadas em identidade, consulte. Exemplos gratuitos de políticas baseadas em identidade RTOS

Políticas baseadas em recursos no Free RTOS

Suporte a políticas baseadas em recursos: não

Políticas baseadas em recursos são documentos JSON de política que você anexa a um recurso. Exemplos de políticas baseadas em recursos são políticas de confiança de IAM funções e políticas de bucket do Amazon S3. Em serviços que suportem políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o recurso ao qual a política está anexada, a política define quais ações um principal especificado pode executar nesse recurso e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. Os diretores podem incluir contas, usuários, funções, usuários federados ou. Serviços da AWS

Para habilitar o acesso entre contas, você pode especificar uma conta ou IAM entidades inteiras em outra conta como principal em uma política baseada em recursos. Adicionar uma entidade principal entre contas à política baseada em recurso é apenas metade da tarefa de estabelecimento da relação de confiança. Quando o principal e o recurso são diferentes Contas da AWS, um IAM administrador na conta confiável também deve conceder permissão à entidade principal (usuário ou função) para acessar o recurso. Eles concedem permissão ao anexar uma política baseada em identidade para a entidade. No entanto, se uma política baseada em recurso conceder acesso a uma entidade principal na mesma conta, nenhuma política baseada em identidade adicional será necessária. Para obter mais informações, consulte Acesso a recursos entre contas IAM no Guia do IAM usuário.

Ações políticas gratuitas RTOS

Compatível com ações de políticas: Sim

Os administradores podem usar AWS JSON políticas para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos, e em que condições.

O `Action` elemento de uma JSON política descreve as ações que você pode usar para permitir ou negar acesso em uma política. As ações de política geralmente têm o mesmo nome da AWS API operação associada. Há algumas exceções, como ações somente com permissão que não têm uma operação correspondente. API Algumas operações também exigem várias ações em uma política. Essas ações adicionais são chamadas de ações dependentes.

Incluem ações em uma política para conceder permissões para executar a operação associada.

Para ver uma lista de RTOS ações gratuitas, consulte Ações definidas por Free RTOS na Referência de autorização de serviço.

As ações de política no Free RTOS usam o seguinte prefixo antes da ação:

```
awes
```

Para especificar várias ações em uma única instrução, separe-as com vírgulas.

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

Para ver exemplos de políticas gratuitas RTOS baseadas em identidade, consulte. Exemplos gratuitos de políticas baseadas em identidade RTOS

Recursos políticos gratuitos RTOS

Compatível com recursos de políticas: Sim

Os administradores podem usar AWS JSON políticas para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos, e em que condições.

O elemento `Resource` JSON de política especifica o objeto ou objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou `NotResource`. Como prática recomendada, especifique um recurso usando seu Amazon Resource Name (ARN). Isso pode ser feito para ações que oferecem compatibilidade com um tipo de recurso específico, conhecido como permissões em nível de recurso.

Para ações que não oferecem compatibilidade com permissões em nível de recurso, como operações de listagem, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

```
"Resource": "*"
```

Para ver uma lista dos tipos de RTOS recursos gratuitos e seusARNs, consulte Recursos definidos por Free RTOS na Referência de autorização de serviço. Para saber com quais ações você pode especificar cada recurso, consulte Ações definidas pelo Free RTOS. ARN

Para ver exemplos de políticas gratuitas RTOS baseadas em identidade, consulte. Exemplos gratuitos de políticas baseadas em identidade RTOS

Chaves de condição de política gratuitas RTOS

Compatível com chaves de condição de política específicas de serviço: Sim

Os administradores podem usar AWS JSON políticas para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos, e em que condições.

O elemento `Condition` (ou bloco `Condition`) permite que você especifique condições nas quais uma instrução estiver em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usem agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação.

Se você especificar vários elementos `Condition` em uma instrução ou várias chaves em um único `Condition` elemento, a AWS os avaliará usando uma operação lógica `AND`. Se você especificar vários valores para uma única chave de condição, AWS avalia a condição usando uma `OR` operação lógica. Todas as condições devem ser atendidas antes que as permissões da instrução sejam concedidas.

Você também pode usar variáveis de espaço reservado ao especificar condições. Por exemplo, você pode conceder permissão a um IAM usuário para acessar um recurso somente se ele estiver marcado com o nome de IAM usuário. Para obter mais informações, consulte elementos de IAM política: variáveis e tags no Guia IAM do usuário.

AWS suporta chaves de condição globais e chaves de condição específicas do serviço. Para ver todas as chaves de condição AWS globais, consulte as chaves de contexto de condição AWS global no Guia IAM do usuário.

Para ver uma lista de chaves de RTOS condição gratuitas, consulte Chaves de condição gratuitas RTOS na Referência de autorização de serviço. Para saber com quais ações e recursos você pode usar uma chave de condição, consulte Ações definidas pelo Free RTOS.

Para ver exemplos de políticas gratuitas RTOS baseadas em identidade, consulte. Exemplos gratuitos de políticas baseadas em identidade RTOS

ACLsem Grátis RTOS

SuportesACLs: Não

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLssão semelhantes às políticas baseadas em recursos, embora não usem o formato de documento JSON de política.

ABACcom grátis RTOS

Suportes ABAC (tags nas políticas): Parciais

O controle de acesso baseado em atributos (ABAC) é uma estratégia de autorização que define permissões com base em atributos. Em AWS, esses atributos são chamados de tags. Você pode anexar tags a IAM entidades (usuários ou funções) e a muitos AWS recursos. Marcar entidades e recursos é a primeira etapa doABAC. Em seguida, você cria ABAC políticas para permitir operações quando a tag do diretor corresponde à tag do recurso que ele está tentando acessar.

ABACé útil em ambientes que estão crescendo rapidamente e ajuda em situações em que o gerenciamento de políticas se torna complicado.

Para controlar o acesso baseado em tags, forneça informações sobre as tags no elemento de condição de uma política usando as `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* ou chaves de condição `aws:TagKeys`.

Se um serviço oferecer suporte às três chaves de condição para cada tipo de recurso, o valor será Sim para o serviço. Se um serviço oferecer suporte às três chaves de condição somente para alguns tipos de recursos, o valor será Parcial

Para obter mais informações sobreABAC, consulte O que éABAC? no Guia do IAM usuário. Para ver um tutorial com etapas de configuraçãoABAC, consulte Usar controle de acesso baseado em atributos (ABAC) no Guia do IAMusuário.

Usando credenciais temporárias com o Free RTOS

Compatível com credenciais temporárias: Sim

Alguns Serviços da AWS não funcionam quando você faz login usando credenciais temporárias. Para obter informações adicionais, incluindo quais Serviços da AWS funcionam com credenciais temporárias, consulte Serviços da AWS esse trabalho IAM no Guia do IAM usuário.

Você está usando credenciais temporárias se fizer login AWS Management Console usando qualquer método, exceto um nome de usuário e senha. Por exemplo, quando você acessa AWS usando o link de login único (SSO) da sua empresa, esse processo cria automaticamente credenciais temporárias. Você também cria automaticamente credenciais temporárias quando faz login no console como usuário e, em seguida, alterna perfis. Para obter mais informações sobre a troca de funções, consulte Alternando para uma função (console) no Guia IAM do usuário.

Você pode criar manualmente credenciais temporárias usando o AWS CLI ou AWS API. Em seguida, você pode usar essas credenciais temporárias para acessar AWS. AWS recomenda que você gere credenciais temporárias dinamicamente em vez de usar chaves de acesso de longo prazo. Para obter mais informações, consulte Credenciais de segurança temporárias emIAM.

Permissões principais entre serviços gratuitas RTOS

Suporta sessões de acesso direto (FAS): Sim

Quando você usa um IAM usuário ou uma função para realizar ações em AWS, você é considerado principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. FASusa as permissões do diretor chamando um Serviço da AWS, combinadas com a solicitação Serviço da AWS para fazer solicitações aos serviços posteriores. FASas solicitações são feitas somente quando um serviço recebe uma solicitação que requer interações com outros Serviços da AWS ou com recursos para ser concluída. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer FAS solicitações, consulte Encaminhar sessões de acesso.

Funções de serviço gratuitas RTOS

Compatível com perfis de serviço: Sim

Uma função de serviço é uma IAMfunção que um serviço assume para realizar ações em seu nome. Um IAM administrador pode criar, modificar e excluir uma função de serviço internamenteIAM. Para obter mais informações, consulte Criação de uma função para delegar permissões a uma Serviço da AWS no Guia do IAM usuário.

> **Warning**
>
> Alterar as permissões de uma função de serviço pode interromper a RTOS funcionalidade gratuita. Edite as funções de serviço somente quando o Free RTOS fornecer orientação para fazer isso.

Funções vinculadas a serviços gratuitamente RTOS

Compatível com perfis vinculados ao serviço: Não

Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. Serviço da AWS O serviço pode presumir a função de executar uma ação em seu nome. As funções vinculadas ao

serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um IAM administrador pode visualizar, mas não editar, as permissões das funções vinculadas ao serviço.

Para obter detalhes sobre como criar ou gerenciar funções vinculadas a serviços, consulte AWS serviços que funcionam com. IAM Encontre um serviço na tabela que inclua um `Yes` na coluna Função vinculada ao serviço. Escolha o link Sim para visualizar a documentação do perfil vinculado a serviço desse serviço.

Exemplos gratuitos de políticas baseadas em identidade RTOS

Por padrão, usuários e funções não têm permissão para criar ou modificar RTOS recursos gratuitos. Eles também não podem realizar tarefas usando o AWS Management Console, AWS Command Line Interface (AWS CLI) ou AWS API. Para conceder permissão aos usuários para realizar ações nos recursos de que precisam, um IAM administrador pode criar IAM políticas. O administrador pode então adicionar as IAM políticas às funções e os usuários podem assumir as funções.

Para saber como criar uma política IAM baseada em identidade usando esses exemplos de documentos de JSON política, consulte Criação de IAM políticas no Guia do IAMusuário.

Para obter detalhes sobre ações e tipos de recursos definidos pelo FreeRTOS, incluindo o formato do ARNs para cada um dos tipos de recursos, consulte Ações, recursos e chaves de condição do Free RTOS na Referência de Autorização de Serviço.

Tópicos

- Melhores práticas de política
- Usando o RTOS console gratuito
- Permitir que usuários visualizem suas próprias permissões

Melhores práticas de política

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir RTOS recursos gratuitos em sua conta. Essas ações podem incorrer em custos para seus Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas AWS gerenciadas e avance para as permissões de privilégios mínimos — Para começar a conceder permissões aos seus usuários e cargas de trabalho, use as políticas AWS gerenciadas que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis no seu Conta da AWS. Recomendamos que você reduza ainda mais as permissões

definindo políticas gerenciadas pelo AWS cliente que sejam específicas para seus casos de uso. Para obter mais informações, consulte políticas AWS gerenciadas ou políticas AWS gerenciadas para funções de trabalho no Guia IAM do usuário.

- Aplique permissões com privilégios mínimos — Ao definir permissões com IAM políticas, conceda somente as permissões necessárias para realizar uma tarefa. Você faz isso definindo as ações que podem ser executadas em atributos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre IAM como usar para aplicar permissões, consulte Políticas e permissões IAM no Guia IAM do usuário.
- Use condições nas IAM políticas para restringir ainda mais o acesso — Você pode adicionar uma condição às suas políticas para limitar o acesso a ações e recursos. Por exemplo, você pode escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usandoSSL. Você também pode usar condições para conceder acesso às ações de serviço se elas forem usadas por meio de uma ação específica Serviço da AWS, como AWS CloudFormation. Para obter mais informações, consulte Elementos IAM JSON da política: Condição no Guia IAM do usuário.
- Use o IAM Access Analyzer para validar suas IAM políticas e garantir permissões seguras e funcionais — o IAM Access Analyzer valida políticas novas e existentes para que as políticas sigam a linguagem da IAM política (JSON) e as melhores práticas. IAM IAMO Access Analyzer fornece mais de 100 verificações de políticas e recomendações práticas para ajudá-lo a criar políticas seguras e funcionais. Para obter mais informações, consulte Validação da política do IAM Access Analyzer no Guia do IAM Usuário.
- Exigir autenticação multifatorial (MFA) — Se você tiver um cenário que exija IAM usuários ou um usuário root Conta da AWS, ative MFA para obter segurança adicional. Para exigir MFA quando API as operações são chamadas, adicione MFA condições às suas políticas. Para obter mais informações, consulte Configurando o API acesso MFA protegido no Guia do IAMusuário.

Para obter mais informações sobre as melhores práticas emIAM, consulte as melhores práticas de segurança IAM no Guia IAM do usuário.

Usando o RTOS console gratuito

Para acessar o RTOS console gratuito, você deve ter um conjunto mínimo de permissões. Essas permissões devem permitir que você liste e visualize detalhes sobre os RTOS recursos gratuitos em seu Conta da AWS. Se você criar uma política baseada em identidade que seja mais restritiva do que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou perfis) com essa política.

Você não precisa permitir permissões mínimas do console para usuários que estão fazendo chamadas somente para AWS CLI o. ou AWS API o. Em vez disso, permita o acesso somente às ações que correspondam à API operação que eles estão tentando realizar.

Para garantir que usuários e funções ainda possam usar o RTOS console gratuito, anexe também a política gratuita RTOS *ConsoleAccess* ou *ReadOnly* AWS gerenciada às entidades. Para obter mais informações, consulte Adicionar permissões a um usuário no Guia do IAM usuário.

Permitir que usuários visualizem suas próprias permissões

Este exemplo mostra como você pode criar uma política que permita IAM aos usuários visualizar as políticas embutidas e gerenciadas que estão anexadas à identidade do usuário. Essa política inclui permissões para concluir essa ação no console ou programaticamente usando o AWS CLI ou. AWS API

```
{
    "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": "*"
        }
    ]
}
```

Solução de problemas de RTOS identidade e acesso gratuitos

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com o Free RTOS e. IAM

Tópicos

- Não estou autorizado a realizar uma ação no Free RTOS
- Não estou autorizado a realizar iam: PassRole
- Quero permitir que pessoas fora da minha Conta da AWS acessem meus RTOS recursos gratuitos

Não estou autorizado a realizar uma ação no Free RTOS

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, é preciso atualizar suas políticas para permitir que você realize a ação.

O exemplo de erro a seguir ocorre quando o `mateojackson` IAM usuário tenta usar o console para ver detalhes sobre um *`my-example-widget`* recurso fictício, mas não tem as permissões fictíciasawes:*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao recurso *`my-example-widget`* usando a ação `awes:`*`GetWidget`*.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Não estou autorizado a realizar iam: PassRole

Se você receber um erro informando que não está autorizado a realizar a `iam:PassRole` ação, suas políticas devem ser atualizadas para permitir que você passe uma função para o FreeRTOS.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazer isso, é preciso ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando um IAM usuário chamado `marymajor` tenta usar o console para realizar uma ação no FreeRTOS. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que pessoas fora da minha Conta da AWS acessem meus RTOS recursos gratuitos

Você pode criar um perfil que os usuários de outras contas ou pessoas fora da sua organização podem usar para acessar seus recursos. Você pode especificar quem é confiável para assumir o perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se o Free é RTOS compatível com esses recursos, consulteComo o Free RTOS funciona com IAM.
- Para saber como fornecer acesso aos seus recursos em todos os Contas da AWS que você possui, consulte Fornecer acesso a um IAM usuário em outro Conta da AWS de sua propriedade no Guia do IAM usuário.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Fornecer Contas da AWS acesso a terceiros no Guia do IAM usuário.
- Para saber como fornecer acesso por meio da federação de identidades, consulte Fornecendo acesso a usuários autenticados externamente (federação de identidades) no Guia do IAMusuário.

- Para saber a diferença entre usar funções e políticas baseadas em recursos para acesso entre contas, consulte Acesso a recursos entre contas IAM no Guia do IAM usuário.

Validação de conformidade

Para saber se um Serviço da AWS está dentro do escopo de programas de conformidade específicos, consulte Serviços da AWS Escopo por Programa de Conformidade Serviços da AWS e escolha o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de AWS conformidade Programas AWS de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixar relatórios em AWS Artifact .

Sua responsabilidade de conformidade ao usar Serviços da AWS é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentações aplicáveis. AWS fornece os seguintes recursos para ajudar na conformidade:

- Guias de início rápido sobre segurança e conformidade — Esses guias de implantação discutem considerações arquitetônicas e fornecem etapas para a implantação de ambientes básicos AWS focados em segurança e conformidade.
- Arquitetura para HIPAA segurança e conformidade na Amazon Web Services — Este whitepaper descreve como as empresas podem usar AWS para criar HIPAA aplicativos qualificados.

> **Note**
>
> Nem todos Serviços da AWS são HIPAA elegíveis. Para obter mais informações, consulte a Referência de serviços HIPAA elegíveis.

- AWS Recursos de https://aws.amazon.com/compliance/resources/ de conformidade — Essa coleção de pastas de trabalho e guias pode ser aplicada ao seu setor e local.
- AWS Guias de conformidade do cliente — Entenda o modelo de responsabilidade compartilhada sob a ótica da conformidade. Os guias resumem as melhores práticas de proteção Serviços da AWS e mapeiam as diretrizes para controles de segurança em várias estruturas (incluindo o Instituto Nacional de Padrões e Tecnologia (NIST), o Conselho de Padrões de Segurança do Setor de Cartões de Pagamento (PCI) e a Organização Internacional de Padronização ()). ISO

- Avaliação de recursos com regras no Guia do AWS Config desenvolvedor — O AWS Config serviço avalia o quão bem suas configurações de recursos estão em conformidade com as práticas internas, as diretrizes e os regulamentos do setor.
- AWS Security Hub— Isso Serviço da AWS fornece uma visão abrangente do seu estado de segurança interno AWS. O Security Hub usa controles de segurança para avaliar os recursos da AWS e verificar a conformidade com os padrões e as práticas recomendadas do setor de segurança. Para obter uma lista dos serviços e controles aceitos, consulte a Referência de controles do Security Hub.
- Amazon GuardDuty — Isso Serviço da AWS detecta possíveis ameaças às suas cargas de trabalho Contas da AWS, contêineres e dados monitorando seu ambiente em busca de atividades suspeitas e maliciosas. GuardDuty pode ajudá-lo a atender a vários requisitos de conformidade, por exemplo PCIDSS, atendendo aos requisitos de detecção de intrusões exigidos por determinadas estruturas de conformidade.
- AWS Audit Manager— Isso Serviço da AWS ajuda você a auditar continuamente seu AWS uso para simplificar a forma como você gerencia o risco e a conformidade com as regulamentações e os padrões do setor.

Resiliência em AWS

A infraestrutura AWS global é construída em torno de AWS regiões e zonas de disponibilidade. AWS As regiões fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, as quais são conectadas com baixa latência, alto throughput e redes altamente redundantes. Com as zonas de disponibilidade, você pode projetar e operar aplicações e bancos de dados que executam o failover automaticamente entre as zonas de disponibilidade sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de datacenter tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte Infraestrutura AWS global.

Segurança de infraestrutura em Free RTOS

AWS os serviços gerenciados são protegidos pelos procedimentos AWS globais de segurança de rede descritos no whitepaper Amazon Web Services: Visão geral dos processos de segurança.

Você usa API chamadas AWS publicadas para acessar AWS serviços pela rede. Os clientes devem oferecer suporte ao Transport Layer Security (TLS) 1.2 ou posterior. Recomendamos TLS 1.3 ou

posterior. Os clientes também devem oferecer suporte a pacotes de criptografia com sigilo direto perfeito (), como Ephemeral Diffie-Hellman (PFS) ou Elliptic Curve Ephemeral Diffie-Hellman (). DHE ECDHE A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando uma ID de chave de acesso e uma chave de acesso secreta associada a um IAM principal. Ou você pode usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Guia de migração do repositório Github do Amazon FreeRTOS

Se você tem um projeto FreeRTOS existente baseado no repositório Amazon FreeRTOS que está preterido, siga estas etapas:

1. Mantenha-se atualizado com as correções de segurança mais recentes e disponíveis ao público. Consulte a página de bibliotecas LTS do FreeRTOS para obter atualizações ou assine o repositório FreeRTOS-LTS no GitHub para receber os patches mais recentes do LTS com correções de bugs críticos e de segurança. Você pode baixar ou clonar os patches de LTS mais recentes do FreeRTOS necessários diretamente dos repositórios individuais do GitHub.
2. Considere refatorar a implementação da interface de transporte de rede para otimizar sua plataforma de hardware. As APIs abstratas, como secure sockets e APIs Wifi não são exigidas pela biblioteca coreMQTT mais recente. Consulte Interface de transporte para obter mais detalhes.

Apêndice

A tabela a seguir fornece recomendações para todos os projetos de demonstração, bibliotecas herdadas e APIs abstratas no repositório do Amazon FreeRTOS.

Bibliotecas e demonstrações migradas

Nome	Type	Recomendações	
coreHTTP	demonstrações e biblioteca	xClone ou baixe a biblioteca coreHTTP diretamente do repositór io coreHTTP (submódulo se estiver usando git) na Organização do GitHub do FreeRTOS. As demonstra ções do coreHTTP estão na distribui ção primária do FreeRTOS. Para obter mais detalhes, consulte a página coreHTTP.	

Nome	Type	Recomendações
coreMQTT	demonstrações e biblioteca	Clone ou baixe a biblioteca coreMQTT diretamente do repositório coreMQTT (submódulo se estiver usando git) na Organização do GitHub do FreeRTOS. As demonstrações do coreMQTT estão na distribuição primária do FreeRTOS. Para obter mais detalhes, consulte a página coreMQTT.
coreMQTT Agent	demonstrações e biblioteca	Clone ou baixe a biblioteca coreMQTT Agent diretamente do repositório coreMQTT Agent (submódulo se estiver usando git) na Organização do GitHub do FreeRTOS. As demonstrações da coreMQTT Agent estão no repositório coreMQTT-Agent-Demos. Para obter mais detalhes, consulte a página coreMQTT-Agent.
device_defender_for_aws	demonstrações e biblioteca	A biblioteca do AWS IoT Device Defender está em seu repositório na organização do GitHub da AWS. Clone ou baixe-o (submódulo se estiver usando git) diretamente do repositório do AWS IoT Device Defender. As demonstrações do AWS IoT Device Defender estão na distribuição primária do FreeRTOS. Para obter mais detalhes, consulte a página AWS IoT Device Defender.

Nome	Type	Recomendações	
device_shadow_for_aws	demonstrações e biblioteca	A biblioteca do AWS IoT Device Defender está em seu repositório na Organização do GitHub da AWS. Clone ou baixe-a (submódulo se estiver usando git) diretamente do repositório do AWS IoT Device Shadow. As demonstrações do AWS IoT Device Shadow estão na distribuição primária do FreeRTOS. Para obter mais detalhes, consulte a página AWS IoT Device Defender.	
jobs_for_aws	demonstrações e biblioteca	A biblioteca Trabalhos do AWS IoT está em seu repositório na Organização do GitHub da AWS. Clone ou baixe-o (submódulo se estiver usando git) diretamente do repositório do Trabalhos do AWS IoT. As demonstrações do Trabalhos do AWS IoT estão na distribuição primária do FreeRTOS. Para obter mais detalhes, consulte a página Trabalhos do AWS IoT.	
OTA	demonstrações e biblioteca	A biblioteca de atualização sem fios do AWS IoT está em seu repositório na Organização do GitHub da AWS. Clone ou baixe-a (submódulo se estiver usando git) diretamente do repositório do OTA AWS IoT. As demonstrações OTA do AWS IoT estão na distribuição primária do FreeRTOS. Para obter mais detalhes, consulte a página OTA AWS IoT.	

Nome	Type	Recomendações
CLI e FreeRTOS_ Plus_CLI	demonstrações e biblioteca	Há um exemplo de CLI em execução no WinSIM. Consulte a página da Interface de Linha de Comando do FreeRTOS Plus para obter mais detalhes. As integrações de referênci a do FreeRTOS IoT em destaque nas plataformas NXP i.MX RT1060 e STM32U5 também fornecem exemplos de CLI em hardware real.
registro em log	Macro	Há implementações da macro de registro em log para plataformas de hardware específicas usadas por algumas das bibliotecas do FreeRTOS. Consulte a página de registro em log para saber como implementar a macro de registro em log. Consulte uma das referências de IoT apresentadas pelo FreeRTOS para ver um exemplo de execução em hardware real.
greengras s_connectivity	Demonstração	[Migração em andamento] Esse projeto de demonstração considero u que a conectividade na nuvem estava disponível antes da conexão com um dispositivo do AWS IoT Greengrass. Um novo projeto que demonstra a capacidade local de autenticação e descoberta está em desenvolvimento. Espere que o novo projeto de demonstração seja publicado em breve na Organização do GitHub do FreeRTOS.

Bibliotecas e demonstrações preteridas

Nome	Type	Recomendações
BLE	demonstrações e biblioteca	A biblioteca BLE do FreeRTOS implementa o protocolo MQTT de proprietário e oferece suporte à publicação e assinatura de tópicos do MQTT por Bluetooth Low Energy (BLE) por meio de um dispositivo de proxy, como um celular. Isso não é mais obrigatório. Use sua própria pilha BLE ou uma opção de terceiros , como o NiMBLE, para otimizar melhor seu projeto.
dev_mode_ key_provisioning	Demonstrações	As integrações de referência do FreeRTOS IoT em destaque nas plataformas NXP i.MX RT1060, STM32U5 ou ESP32-C3 fornecem exemplos de provisionamento crucial usando uma CLI.
posix	abstração e demonstração	Não recomendado para uso.
wifi_provisioning	exemplo	Este exemplo demonstrou como provisionar credenciais WiFi em um dispositivo usando a biblioteca BLE do Amazon FreeRTOS. Consulte a referência de IoT em destaque do FreeRTOS na plataforma ESP32C3 para obter um exemplo de provision amento WiFi via BLE.
APIs abstratas herdadas	código	Essas são APIs criadas para fornecer uma interface abstrata para várias pilhas de software, módulos

Nome	Type	Recomendações	
		de conectividade e plataformas MCU de terceiros de vários fornecedores. Por exemplo, existem interfaces para abstração de WiFi, soquetes seguros e assim por diante. Há suporte para eles no repositório Amazon-FreeRTOS e isso está na pasta `/libraries/abstractions/` . Essas APIs não são necessárias ao usar as bibliotecas LTS do FreeRTOS.	

As bibliotecas e demonstrações na tabela acima não receberão patches de segurança ou correções de bugs.

Bibliotecas de terceiros

Quando as demonstrações no Amazon-FreeRTOS usam bibliotecas de terceiros, recomendamos que você as submodule diretamente de seus repositórios de terceiros.

- CMock: clone (submódulo se você usa git) diretamente do repositório Cmock.
- jsmn: não recomendado e não há mais suporte para ela.
- lwip: clone (submódulo se você usa git) diretamente do repositório lwip-tcpip.
- lwip_osal: consulte as Integrações de Referência em destaque do FreeRTOS no i.MX RT1060 ou STM32U5 para saber como implementar o lwip_osal em sua placa/plataforma de hardware.
- d: clone (submódulo se você usa git) diretamente do repositório Mbed-TLS. A configuração e os utilitários do mbedtls podem ser reutilizados; nesse caso, faça uma cópia local.
- pkcs11: clone-o (submódulo se você usar git) diretamente da biblioteca corePKCS11 ou do repositório OASIS PKCS 11.
- tinycbor: clone (submódulo se você usa git) diretamente do repositório tinycbor.
- tinycrypt: recomendamos que você use aceleradores de criptografia da sua plataforma MCU, se disponíveis. Se você quiser continuar usando o tinycrypt, clone-o (submódulo se você usar git) diretamente do repositório tinycrypt.

- tracealyzer_recorder: clone-o (submódulo se você usar git) diretamente do repositório de gravadores de rastreamento do trace recorder.
- unity: clone-o (submódulo se você usar git) diretamente do repositório ThrowTheSwitch/Unity.
- win_pcap : a win_pcap não é mais mantida. Recomendamos que você use libslirp, libpcap (posix) ou npcap.

Testes de portabilidade e testes de integração

Todos os testes da pasta `/tests` necessários para validar a integração das bibliotecas do FreeRTOS foram migrados para o repositório FreeRTOS-Libraries-Integration-Tests. Eles podem ser usados para testar a implementação do PAL e a integração da biblioteca. Os mesmos testes são usados pelo AWS IoT Device Tester (IDT) para o Programa de qualificação de dispositivos da AWS para FreeRTOS.

Documentação arquivada do FreeRTOS

Arquivo do Guia do usuário do FreeRTOS

Essas versões anteriores do Guia do usuário do FreeRTOS estão disponíveis para uso com as versões LTS (suporte de longo prazo) do FreeRTOS.

- Guia do usuário do FreeRTOS para a versão 202210.00 do FreeRTOS
- Guia do usuário do FreeRTOS para a versão 202012.00 do FreeRTOS

Conteúdo do Guia do usuário do FreeRTOS anterior

Esse conteúdo está obsoleto, mas é fornecido aqui para referência.

Consulte Conceitos básicos do FreeRTOS para obter os links para conteúdos recentes.

Conceitos básicos do FreeRTOS

> **Important**
>
> Essa página se refere ao repositório do Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial de Conceitos básicos do FreeRTOS mostra como fazer download e configurar o FreeRTOS em uma máquina de host e, em seguida, compilar e executar um aplicativo de demonstração simples em uma placa de microcontrolador qualificada.

Por meio deste tutorial, supomos que você esteja familiarizado com a AWS IoT e o console da AWS IoT. Caso contrário, recomendamos que você conclua o tutorial de Conceitos básicos da AWS IoT antes de continuar.

Tópicos

- Aplicativo de demonstração do FreeRTOS

- Primeiras etapas
- Solução de problemas de conceitos básicos
- Uso da CMake com o FreeRTOS
- Provisionamento de chaves no modo de desenvolvedor
- Guias de conceitos básicos específicos da placa
- Próximas etapas com o FreeRTOS

Aplicativo de demonstração do FreeRTOS

> **Important**
>
> Essa página se refere ao repositório do Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

O aplicativo de demonstração usado neste tutorial é a demonstração da coreMQTT Agent, definido no arquivo *freertos*`/demos/coreMQTT_Agent/mqtt_agent_task.c`. Ele usa Biblioteca coreMQTT para se conectar à nuvem da AWS e depois publica mensagens periodicamente em um tópico MQTT hospedado pelo agente MQTT do AWS IoT.

Apenas um aplicativo de demonstração do FreeRTOS pode ser executado por vez. Quando um projeto de demonstração do FreeRTOS é compilado, a primeira demonstração habilitada no arquivo de cabeçalho *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/aws_demo_config.h` é o aplicativo em execução. Para este tutorial, você não precisa habilitar ou desabilitar as demonstrações. A demonstração da coreMQTT Agent está habilitada por padrão.

Para obter mais informações sobre os aplicativos de demonstração incluídos com o FreeRTOS, consulte Demonstrações do FreeRTOS.

Primeiras etapas

> **Important**
>
> Essa página se refere ao repositório do Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto

FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Para começar a usar o FreeRTOS AWS IoT com, você deve ter AWS uma conta, um usuário com permissões de acesso AWS IoT e serviços em nuvem do FreeRTOS. Você também deve baixar o FreeRTOS e configurar o projeto de demonstração do FreeRTOS da sua placa para trabalhar com ele. AWS IoT As seções a seguir o orientam ao longo desses requisitos.

Note

- Se você estiver usando o Espressif DevKit ESP32-C, o ESP-WROVER-KIT ou o ESP32-WROOM-32SE, pule essas etapas e vá para. Introdução ao Espressif ESP32- DevKit C e ao ESP-WROVER-KIT
- Se estiver usando o Nordic nRF52840-DK, ignore estas etapas e vá para Conceitos básicos do Nordic nRF52840-DK.

1. Configurando sua AWS conta e permissões
2. Registrando sua placa MCU com AWS IoT
3. Fazer download do FreeRTOS
4. Configuração das demonstrações do FreeRTOS

Configurando sua AWS conta e permissões

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

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

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

Para conceder acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos em AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Criação de um conjunto de permissões no Guia do usuário do AWS IAM Identity Center .
- Usuários gerenciados no IAM com provedor de identidades:

  Crie um perfil para a federação de identidades. Siga as instruções em Criar um perfil para um provedor de identidades de terceiros (federação) no Guia do usuário do IAM.
- Usuários do IAM:
  - Crie um perfil que seu usuário possa assumir. Siga as instruções em Criação de um perfil para um usuário do IAM no Guia do usuário do IAM.
  - (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Registrando sua placa MCU com AWS IoT

Seu conselho deve estar registrado AWS IoT para se comunicar com a AWS nuvem. Para registrar seu quadro com AWS IoT, você deve ter:

Uma AWS IoT política

A AWS IoT política concede ao seu dispositivo permissões para acessar AWS IoT recursos. Ele é armazenado na AWS nuvem.

Qualquer AWS IoT coisa

Qualquer AWS IoT coisa permite que você gerencie seus dispositivos em AWS IoT. Ele é armazenado na AWS nuvem.

Uma chave privada e um certificado X.509

A chave privada e o certificado permitem que seu dispositivo se autentique com AWS IoT.

Para registrar a placa manualmente, siga os procedimentos abaixo.

Para criar uma AWS IoT política

1. Para criar uma política do IAM, você precisa saber sua AWS região e o número AWS da conta.

   Para encontrar o número da sua AWS conta, abra o AWS Management Console, localize e expanda o menu abaixo do nome da sua conta no canto superior direito e escolha Minha conta. O ID da conta é exibido em Account Settings (Configurações da conta).

   Para encontrar a AWS região da sua AWS conta, use AWS Command Line Interface o. Para instalar o AWS CLI, siga as instruções no Guia do AWS Command Line Interface usuário. Depois de instalar o AWS CLI, abra uma janela do prompt de comando e digite o seguinte comando:

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

   O resultado deve ser semelhante ao seguinte:

   ```
   {
       "endpointAddress": "xxxxxxxxxxxxxx-ats.iot.us-west-2.amazonaws.com"
   }
   ```

Neste exemplo, a região é `us-west-2`.

> **Note**
>
> Recomendamos o uso de endpoints do ATS, conforme mostrado no exemplo.

2. Navegue até o console do AWS IoT.
3. No painel de navegação, escolha Secure (Seguro), Policies (Políticas) e Create (Criar).
4. Insira um nome para identificar a política.
5. Na seção Add statements (Adicionar instruções), escolha Advanced mode (Modo avançado). Copie e cole o seguinte JSON na janela do editor de política. Substitua *aws-region* e *aws-account* por sua AWS região e ID da conta.

```
{
    "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:*"
    }
    ]
}
```

Essa política concede as seguintes permissões:

`iot:Connect`

Concede ao seu dispositivo a permissão para se conectar ao agente de AWS IoT mensagens com qualquer ID de cliente.

`iot:Publish`

Concede ao dispositivo permissão para publicar uma mensagem MQTT em qualquer tópico MQTT.

`iot:Subscribe`

Concede ao dispositivo permissão para assinar o filtro de tópico MQTT.

`iot:Receive`

Concede ao dispositivo permissão para receber mensagens do agente de mensagem da AWS IoT em qualquer tópico MQTT.

6. Escolha Criar.

Para criar uma coisa do IoT, chave privada e certificado para o dispositivo

1. Navegue até o console do AWS IoT.
2. No painel de navegação, escolha Manage (Gerenciar) e, depois, Things (Coisas).
3. Se você não tiver nada do IoT registrado na sua conta, a página You don't have any things yet (Você não tem coisas ainda) será exibida. Se você visualizar esta página, selecione Register a thing (Registrar uma coisa). Caso contrário, escolha Criar.
4. Na página Criando AWS IoT coisas, escolha Criar uma única coisa.
5. Na página Add your device to the thing registry (Adicionar seu dispositivo ao registro de coisas), insira um nome para a coisa e escolha Next (Próximo).
6. Na página Add a certificate for your thing (Adicionar um certificado à sua coisa), em One-click certificate creation (Criação do certificado com um clique, escolha Create certificate (Criar certificado).
7. Baixe a chave privada e o certificado escolhendo os links Download para cada um deles.
8. Selecione Activate (Ativar) para ativar o certificado. Os certificados devem ser ativados antes do uso.
9. Escolha Anexar uma política para anexar uma política ao seu certificado que conceda ao seu dispositivo acesso às AWS IoT operações.

10. Escolha a política que você acabou de criar e escolha Register thing (Registrar coisa).

Depois que seu conselho for registrado AWS IoT, você poderá continuarFazer download do FreeRTOS.

Fazer download do FreeRTOS

Você pode baixar o FreeRTOS do repositório do FreeRTOS. GitHub

Depois de fazer download do FreeRTOS, você poderá prosseguir para Configuração das demonstrações do FreeRTOS.

Configuração das demonstrações do FreeRTOS

É necessário editar alguns arquivos de configuração no diretório do FreeRTOS antes de compilar e executar demonstrações na placa.

Para configurar seu AWS IoT endpoint

Você deve fornecer ao FreeRTOS AWS IoT seu endpoint para que o aplicativo executado em sua placa possa enviar solicitações para o endpoint correto.

1. Navegue até o console do AWS IoT.
2. No painel de navegação à esquerda, escolha Configurações.

   Seu AWS IoT endpoint é exibido em Device data endpoint. Deve ser semelhante a *1234567890123*-ats.iot.*us-east-1*.amazonaws.com. Anote esse endpoint.
3. No painel de navegação, escolha Manage (Gerenciar) e, depois, Things (Coisas).

   Seu dispositivo deve ter um nome de AWS IoT coisa. Anote esse nome.
4. Abra o demos/include/aws_clientcredential.h.
5. Especifique valores para as seguintes constantes:

   - #define clientcredentialMQTT_BROKER_ENDPOINT "*Your AWS IoT endpoint*";
   - #define clientcredentialIOT_THING_NAME "*The AWS IoT thing name of your board*"

Para configurar a rede Wi-Fi

Se a sua placa estiver conectada à Internet por meio de uma conexão Wi-Fi, será necessário fornecer ao FreeRTOS as credenciais de Wi-Fi para se conectar à rede. Se a placa não oferecer suporte a Wi-Fi, você poderá ignorar estas etapas.

1. `demos/include/aws_clientcredential.h`.
2. Especifique os valores para as seguintes constantes `#define`:

   - `#define clientcredentialWIFI_SSID "`*`The SSID for your Wi-Fi network`*`"`
   - `#define clientcredentialWIFI_PASSWORD "`*`The password for your Wi-Fi network`*`"`
   - `#define clientcredentialWIFI_SECURITY` *`O tipo de segurança da rede Wi-Fi`*

     Os tipos de segurança válidos são:

     - `eWiFiSecurityOpen` (Aberto, sem segurança)
     - `eWiFiSecurityWEP` (segurança WEP)
     - `eWiFiSecurityWPA` (segurança WPA)
     - `eWiFiSecurityWPA2` (segurança WPA2)

Para formatar suas AWS IoT credenciais

Os FreeRTOS devem ter AWS IoT o certificado e as chaves privadas associados à sua coisa registrada e suas políticas de permissões para se comunicar AWS IoT com sucesso em nome do seu dispositivo.

> **Note**
>
> Para configurar suas AWS IoT credenciais, você deve ter a chave privada e o certificado que você baixou do AWS IoT console quando registrou seu dispositivo. Depois de registrar seu dispositivo como uma AWS IoT coisa, você pode recuperar certificados de dispositivo do AWS IoT console, mas não pode recuperar chaves privadas.

O FreeRTOS é um projeto de linguagem C e o certificado e a chave privada devem ser formatados especialmente para serem adicionados ao projeto.

1. Em uma janela do navegador, abra `tools/certificate_configuration/CertificateConfigurator.html`.
2. Em Arquivo PEM do certificado, selecione *ID*`-certificate.pem.crt` do qual foi feito download no console do AWS IoT .
3. Em Arquivo PEM da chave privada, selecione *ID*`-private.pem.key` do qual foi feito download no console do AWS IoT .
4. Selecione Generate and save aws_clientcredential_keys.h (Gerar e salvar aws_clientcredential_keys.h) e salve o arquivo em `demos/include`. Isso substitui o arquivo existente no diretório.

> **Note**
>
> O certificado e a chave privada são codificados para fins de demonstração somente. Por este motivo, as aplicações devem armazenar esses arquivos em um local seguro.

Após configurar o FreeRTOS, você pode prosseguir no guia de conceitos básicos para sua placa configurar o hardware da plataforma e seu ambiente de desenvolvimento de software e, em seguida, compilar e executar a demonstração na sua placa. Para obter instruções específicas da placa, consulte o Guias de conceitos básicos específicos da placa. A aplicação de demonstração usada no tutorial de Conceitos básicos é a demonstração de autenticação mútua coreMQTT, localizada em `demos/coreMQTT/mqtt_demo_mutual_auth.c`.

Solução de problemas de conceitos básicos

> **Important**
>
> Essa página se refere ao repositório do Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Os tópicos a seguir podem ajudá-lo a solucionar problemas encontrados ao começar a usar o FreeRTOS:

Tópicos

- Dicas gerais da solução de problemas de conceitos básicos
- Instalação de um emulador de terminal

Para obter a solução de problemas específicas da placa, consulte o guia Conceitos básicos do FreeRTOS da placa.

Dicas gerais da solução de problemas de conceitos básicos

Nenhuma mensagem é exibida no console do AWS IoT após executar o projeto de demonstração Olá, mundo. O que devo fazer?

Faça o seguinte:

1. Abra uma janela de terminal para visualizar a saída do registro da amostra. Isso pode ajudar você a determinar o que está errado.
2. Verifique se suas credenciais de rede são válidas.

Os logs mostrados no meu terminal ao executar uma demonstração estão truncados. Como posso aumentar comprimento deles?

Aumente o valor de `configLOGGING_MAX_MESSAGE_LENGTH` a 255 no arquivo `FreeRTOSconfig.h` da demonstração que você está executando:

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

Instalação de um emulador de terminal

Um emulador de terminal pode ajudar você a diagnosticar problemas ou verificar se o código do dispositivo está sendo executado corretamente. Há uma variedade de emuladores de terminal disponíveis para Windows, macOS e Linux.

É necessário conectar a placa ao computador antes de tentar estabelecer uma conexão serial com sua placa usando um emulador de terminal.

Use as seguintes configurações para definir o emulador de terminal:

Configuração do terminal	Valor
Taxa de BAUD	115200
Dados	8 bits
Paridade	none
Parar	1 bit
Controle de fluxo	none

Localização da porta serial da placa

Se você não souber qual é a portal serial da sua placa, poderá emitir um dos seguintes comandos da linha de comando ou do terminal para retornar as portas seriais de todos os dispositivos conectados ao computador host:

Windows

```
chgport
```

Linux

```
ls /dev/tty*
```

macOS

```
ls /dev/cu.*
```

Uso da CMake com o FreeRTOS

Important

Essa página se refere ao repositório do Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto

FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Você pode usar CMake para gerar arquivos de compilação do projeto a partir de código-fonte da aplicação FreeRTOS e compilar e executar o código-fonte.

Você também pode usar um IDE para editar, depurar, compilar, atualizar e executar código em dispositivos qualificados para o FreeRTOS. Cada guia de conceitos básicos específico para a placa inclui instruções para configurar o IDE para uma plataforma específica. Se você preferir trabalhar sem um IDE, você pode usar outras ferramentas de terceiros de edição de código e depuração para desenvolver e depurar seu código e, então, usar CMake para compilar e executar os aplicativos.

As seguintes placas são compatíveis com CMake:

- Espressif ESP32- C DevKit
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT Connectivity Kit
- Kit inicial Marvell MW320 AWS IoT
- Kit inicial Marvell MW322 AWS IoT
- Pacote Microchip Curiosity PIC32MZEF
- Kit de desenvolvimento Nordic nRF52840 DK
- Nó de IoT do kit STMicroelectronicsSTM32L4 Discovery
- Texas Instruments CC3220SF-LAUNCHXL
- Simulador do Microsoft Windows

Consulte os tópicos a seguir para obter mais informações sobre como usar CMake com FreeRTOS.

Tópicos

- Pré-requisitos
- Desenvolvimento de aplicações do FreeRTOS com ferramentas de terceiros para edição e depuração de código
- Compilação do FreeRTOS com o CMake

Pré-requisitos

Certifique-se de que sua máquina host atenda aos seguintes pré-requisitos antes de continuar:

- O conjunto de ferramentas de compilação do seu dispositivo deve ser compatível com o sistema operacional da máquina. O CMake é compatível com todas as versões do Windows, macOS e Linux

  O subsistema do Windows para Linux (WSL) não é compatível. Use o CMake nativo em máquinas Windows.
- É necessário ter o CMake versão 3.13 ou posterior instalado.

  Você pode fazer download da distribuição binária do CMake em CMake.org.

  > **Note**
  >
  > Se você fizer download da distribuição binária do CMake, certifique-se de adicionar o executável CMake à variável de ambiente PATH antes de usar o CMake na linha de comando.

  Você também pode fazer download e instalar o CMake usando um gerenciador de pacotes, como homebrew no macOS e scoop ou chocolatey no Windows.

  > **Note**
  >
  > As versões do pacote CMake fornecidas nos gerenciadores de pacotes para muitas distribuições Linux são. out-of-date Se o gerenciador de pacote de distribuição não fornecer a versão mais recente do CMake, experimente gerenciadores de pacote alternativos, como `linuxbrew` ou `nix`.
- Você deve ter um sistema de compilação nativo compatível.

  O CMake pode direcionar muitos sistemas de compilação nativos, inclusive o GNU Make ou o Ninja. Tanto o Make quanto o Ninja podem ser instalados com gerenciadores de pacotes no Linux, no macOS e no Windows. Se você estiver usando o Make no Windows, poderá instalar uma versão independente do Equation ou instalar o MinGW, que inclui o Make no pacote.

> **Note**
>
> O executável Make no MinGW é chamado `mingw32-make.exe`, em vez de `make.exe`.

Recomendamos que você use o Ninja, porque ele é mais rápido do que o Make e também fornece suporte nativo a todos os sistemas operacionais de desktop.

Desenvolvimento de aplicações do FreeRTOS com ferramentas de terceiros para edição e depuração de código

Você pode usar um editor de código e uma extensão de depuração ou uma ferramenta de terceiros de depuração para desenvolver aplicações para o FreeRTOS.

Se, por exemplo, você usar o Visual Studio Code como editor de código, poderá instalar a extensão Cortex-Debug VS Code como depurador. Ao concluir o desenvolvimento de seu aplicativo, você pode invocar a ferramenta de linha de comando CMake para criar seu projeto no VS Code. Para obter mais informações sobre como usar CMake para compilar aplicações do FreeRTOS, consulte Compilação do FreeRTOS com o CMake.

Para a depuração, você pode fornecer um VS Code com configuração de depuração semelhante ao seguinte:

```
"configurations": [
    {
        "name": "Cortex Debug",
        "cwd": "${workspaceRoot}",
        "executable": "./build/st/stm32l475_discovery/aws_demos.elf",
        "request": "launch",
        "type": "cortex-debug",
        "servertype": "stutil"
    }
]
```

Compilação do FreeRTOS com o CMake

O CMake seleciona seu sistema operacional host como o sistema de destino por padrão. Para usá-lo em compilação cruzada, o CMake exige um arquivo de conjunto de ferramentas, que especifica o compilador que você deseja usar. No FreeRTOS, fornecemos arquivos de cadeia de ferramentas

padrão no *freertos*/tools/cmake/toolchains. A maneira de fornecer esse arquivo para o CMake depende de você estar usando a interface de linha de comando ou a GUI do CMake. Para obter mais detalhes, siga as instruções de Geração de arquivos de compilação (ferramenta de linha de comando do CMake) abaixo. Para obter mais informações sobre compilação cruzada no CMake, consulte CrossCompilingo wiki oficial do CMake.

Para criar um projeto com base no CMake

1. Execute o CMake para gerar os arquivos de compilação para um sistema de compilação nativo, como o Make ou o Ninja.

   É possível usar a ferramenta de linha de comando do CMake ou a GUI do CMake para gerar os arquivos de compilação para seu sistema de compilação nativo.

   Para obter informações sobre como gerar arquivos de compilação do FreeRTOS, consulte Geração de arquivos de compilação (ferramenta de linha de comando do CMake) e Geração de arquivos de compilação (GUI do CMake).
2. Invoque o sistema de compilação nativo para transformar o projeto em um executável.

   Para obter informações sobre como criar arquivos de compilação do FreeRTOS, consulte Compilação do FreeRTOS a partir de arquivos de compilação gerados.

Geração de arquivos de compilação (ferramenta de linha de comando do CMake)

Você pode usar a ferramenta de linha de comando CMake para gerar arquivos de compilação do FreeRTOS. Para gerar os arquivos de compilação, especifique uma placa de destino, um compilador e o local do código-fonte e o diretório de compilação.

É possível usar as seguintes opções para o cmake:

- -DVENDOR: especifica a placa de destino.
- -DCOMPILER: especifica o compilador.
- -S: especifica a localização do código-fonte.
- -B: especifica a localização dos arquivos de compilação gerados.

> **Note**
>
> O compilador deve estar na variável PATH do sistema ou você deve especificar o local do compilador.

Por exemplo, se o fornecedor for a Texas Instruments, a placa for Launchpad CC3220 e o compilador for GCC para ARM, você poderá emitir o seguinte comando para compilar os arquivos de origem do diretório atual em um diretório denominado *build-directory*:

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory
```

> **Note**
>
> Se você estiver usando o Windows, deverá especificar o sistema de compilação nativo, pois, o CMake usa o Visual Studio por padrão. Por exemplo: .
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G Ninja
> ```
>
> Ou:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G "MinGW Makefiles"
> ```

As expressões regulares `${VENDOR}.*` e `${BOARD}.*` são usadas para pesquisar uma placa correspondente, portanto, você não precisa usar os nomes completos do fornecedor e da placa para as opções `BOARD` e `VENDOR`. Nomes parciais funcionam, desde que haja uma única correspondência. Por exemplo, os seguintes comandos geram os mesmos arquivos de compilação da mesma origem:

```
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
```

Você pode usar a opção `CMAKE_TOOLCHAIN_FILE` se quiser usar um arquivo de conjunto de ferramentas que não esteja localizado no diretório padrão `cmake/toolchains`. Por exemplo: .

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -S . -
B build-directory
```

Se o arquivo de conjunto de ferramentas não usar caminhos absolutos para o compilador e você não tiver adicionado o compilador à variável de ambiente de `PATH`, talvez o CMake não conseguirá encontrá-lo. Para certificar-se de que o CMake encontre seu arquivo de conjunto de ferramentas, você pode usar a opção `AFR_TOOLCHAIN_PATH`. Essa opção pesquisa o caminho do diretório do conjunto de ferramentas especificado e a subpasta do conjunto de ferramentas em `bin`. Por exemplo: .

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -
DAFR_TOOLCHAIN_PATH='/path/to/toolchain/' -S . -B build-directory
```

Para habilitar a depuração, defina `CMAKE_BUILD_TYPE` como `debug`. Com essa opção ativada, o CMake adiciona sinalizadores de depuração às opções de compilação e cria o FreeRTOS com símbolos de depuração.

```
# Build with debug symbols
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -DCMAKE_BUILD_TYPE=debug -S . -B build-directory
```

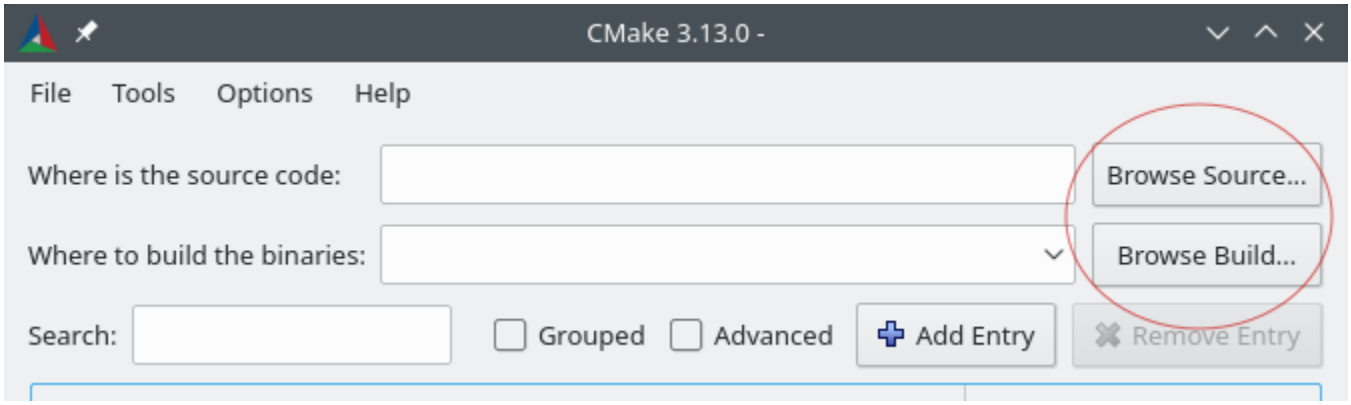
Você também pode definir o `CMAKE_BUILD_TYPE` como `release` para adicionar sinalizadores de otimização às opções de compilação.

Geração de arquivos de compilação (GUI do CMake)

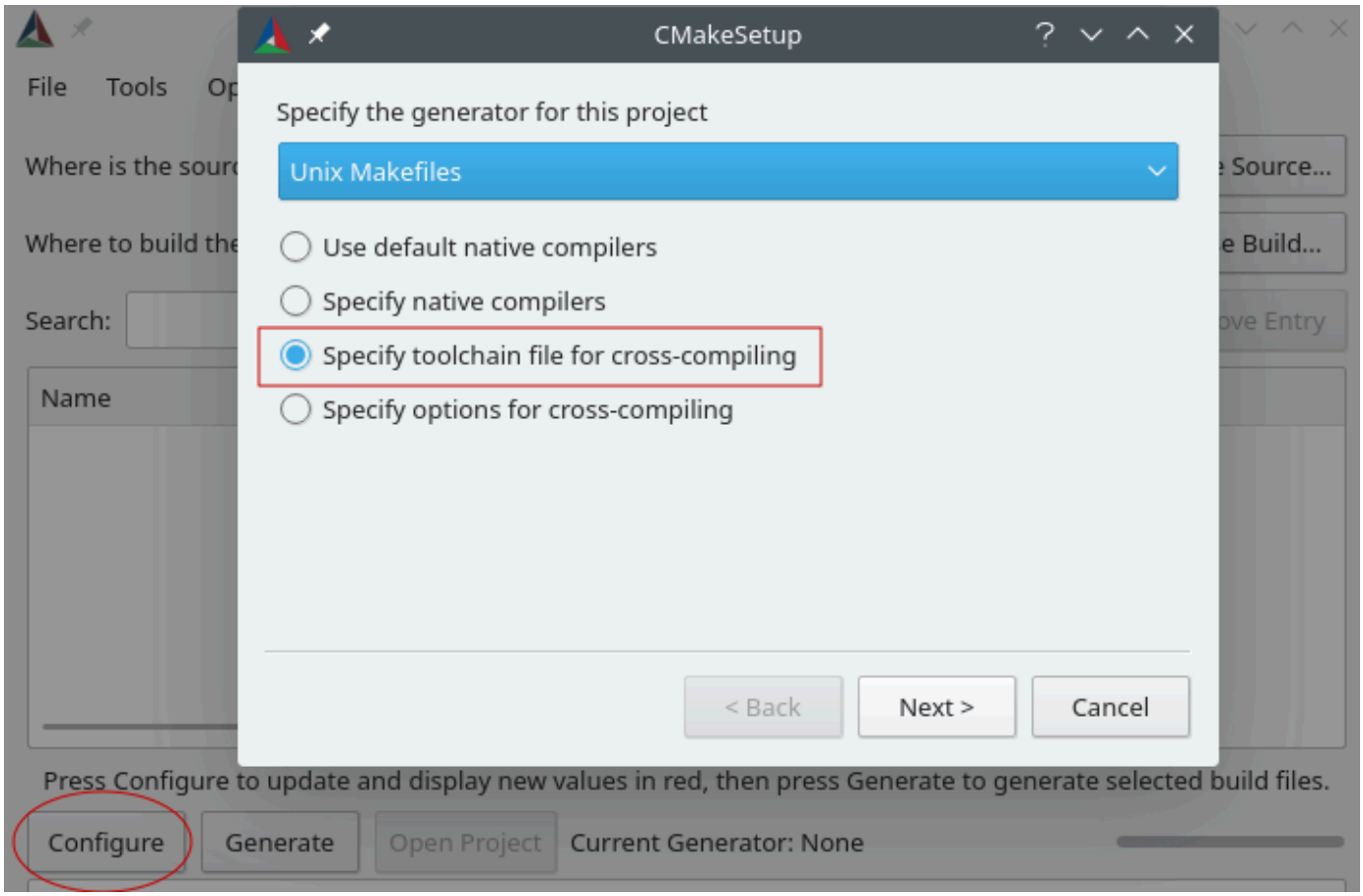
Você pode usar a GUI do CMake para gerar arquivos de compilação do FreeRTOS.

Para gerar arquivos de compilação com a GUI do CMake

1. Na linha de comando, emita `cmake-gui` para iniciar a GUI.
2. Escolha Browse Source (Procurar origem) e especifique a entrada de origem e escolha Browse Build (Procurar compilação) e especifique a saída da compilação.



3. Escolha Configure (Configurar) e, em Specify the build generator for this project (Especificar o gerador de compilação para esse projeto), localize e escolha o sistema de compilação que deseja usar para compilar os arquivos de compilação gerados. Se você não vir a janela pop-up, pode reutilizar um diretório de compilação existente. Neste caso, exclua o cache do CMake escolhendo Delete Cache (Excluir cache) no menu File (Arquivo).



4. Escolha Specify toolchain file for cross-compiling (Especificar arquivo de conjunto de ferramentas para compilação cruzada) e escolha Next (Próximo).

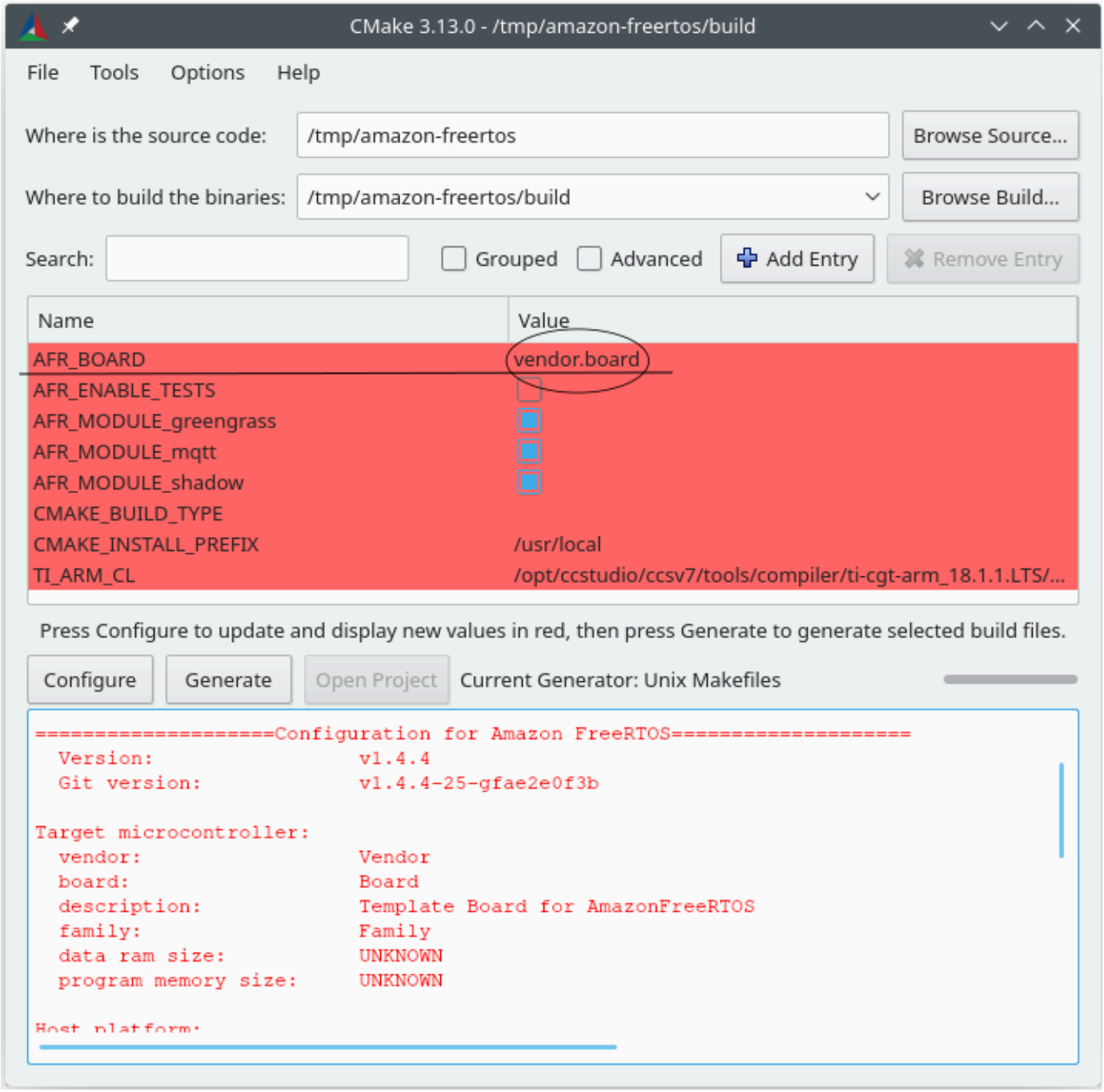
5. Escolha o arquivo de cadeia de ferramentas (por exemplo, *freertos*/tools/cmake/toolchains/arm-ti.cmake) e escolha Concluir.

   A configuração padrão do FreeRTOS é a placa de modelo, que não fornece nenhum destino de camada portátil. Como resultado, uma janela será exibida com a mensagem Error in configuration process.

   > **Note**
   >
   > Se você estiver vendo o seguinte erro:
   >
   > ```
   > CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):
   > Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
   > ```

   Isso significa que o compilador não está na sua variável de ambiente PATH. Você pode definir a variável AFR_TOOLCHAIN_PATH na GUI para informar o CMake onde você instalou o compilador. Se você não vir a variável AFR_TOOLCHAIN_PATH, escolha Add Entry (Adicionar entrada). Na janela pop-up, em Name (Nome), digite **AFR_TOOLCHAIN_PATH**. Em Compiler Path (Caminho do compilador), digite o caminho para o compilador. Por exemplo, C:/toolchains/arm-none-eabi-gcc.

6. Agora, a GUI deve ter a seguinte aparência:



Escolha AFR_BOARD, escolha a placa eConfigure (Configurar) novamente.

7. Escolha Gerar. O CMake gera os arquivos do sistema de compilação (por exemplo, makefiles ou arquivos ninja), e esses arquivos aparecem no diretório de compilação que você especificou na primeira etapa. Siga as instruções na próxima seção para gerar a imagem binária.

Compilação do FreeRTOS a partir de arquivos de compilação gerados

Criação com sistema de compilação nativo

Você pode compilar o FreeRTOS com um sistema de compilação nativo chamando o comando de sistema de compilação a partir do diretório de binários de saída.

Por exemplo, se o diretório de saída do arquivo de compilação for <build_dir> e você estiver usando o Make como seu sistema de compilação nativo, execute os seguintes comandos:

```
cd <build_dir>
make -j4
```

Criar o com o CMake

Você também pode usar a ferramenta de linha de comando do CMake para criar o FreeRTOS. O CMake fornece uma camada de abstração para chamar os sistemas de compilação nativos. Por exemplo: .

```
cmake --build build_dir
```

Estes são alguns outros usos comuns do modo de compilação da ferramenta de linha de comando do 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 obter mais informações sobre o modo de compilação do CMake, consulte Documentação do CMake.

Provisionamento de chaves no modo de desenvolvedor

> **Important**
>
> Essa página se refere ao repositório do Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

Esta seção discute duas opções para obter um certificado de cliente X.509 confiável em um dispositivo IoT para testes de laboratório. Dependendo dos recursos do dispositivo, várias operações relacionadas ao provisionamento podem ou não ser compatíveis, incluindo geração de chaves ECDSA integradas, importação de chaves privadas e registro de certificados X.509. Além disso, diferentes casos de uso exigem diferentes níveis de proteção de chaves, desde o armazenamento flash integrado até o uso de hardware criptográfico dedicado. Esta seção fornece lógica para trabalhar nos recursos criptográficos do seu dispositivo.

Opção 1: importação de chaves privadas da AWS IoT

Para fins de testes laboratoriais, se o seu dispositivo permitir a importação de chaves privadas, siga as instruções em Configuração das demonstrações do FreeRTOS.

Opção 2: geração de chaves privadas integradas

Se o seu dispositivo tiver um elemento seguro, ou se preferir gerar o seu próprio par de chaves de dispositivo e certificado, siga estas instruções.

Configuração inicial

Primeiro, execute as etapas em Configuração das demonstrações do FreeRTOS, mas ignore a última etapa (ou seja, ignore Como formatar suas credenciais da AWS IoT). O resultado líquido deve ser que o arquivo `demos/include/aws_clientcredential.h` foi atualizado com suas configurações, mas o arquivo `demos/include/aws_clientcredential_keys.h` não.

Configuração do projeto de demonstração

Abra a demonstração da autenticação mútua do coreMQTT conforme descrito no guia da sua placa em Guias de conceitos básicos específicos da placa. No projeto,

abra o arquivo `aws_dev_mode_key_provisioning.c` e altere a definição de `keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR`, definido como zero por padrão, para um:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

Depois, compile e execute o projeto de demonstração e continue na próxima etapa.

Extração de chaves públicas

Como o dispositivo não foi provisionado com uma chave privada e um certificado de cliente, a demonstração não será autenticada no AWS IoT. No entanto, a demonstração de autenticação mútua coreMQTT começa executando o provisionamento de chaves em modo de desenvolvedor, resultando na criação de uma chave privada se ainda não estiver presente. Você verá algo parecido com o seguinte perto do início da saída do console serial.

```
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 as seis linhas de bytes de chave em um arquivo chamado `DevicePublicKeyAsciiHex.txt`. Depois, use a ferramenta de linha de comando "xxd" para analisar os bytes hexadecimais em binário:

```
xxd -r -ps DevicePublicKeyAsciiHex.txt DevicePublicKeyDer.bin
```

Use "openssl" para formatar a chave pública do dispositivo codificado binário (DER) como PEM:

```
openssl ec -inform der -in DevicePublicKeyDer.bin -pubin -pubout -outform pem -out
 DevicePublicKey.pem
```

Não se esqueça de desativar a configuração de geração de chaves temporária ativada acima. Caso contrário, o dispositivo criará mais um par de chaves, e será necessário repetir as etapas anteriores:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

Configuração da infraestrutura de chave pública

Siga as instruções em Registrar seu certificado de CA a fim de criar uma hierarquia de certificados para seu certificado de laboratório de dispositivos. Pare antes de executar a sequência descrita na seção Criar um certificado de dispositivo usando seu certificado CA.

Nesse caso, o dispositivo não assinará a solicitação de certificado (ou seja, a solicitação de serviço de certificado ou CSR) porque a lógica de codificação X.509 necessária para criar e assinar uma CSR foi excluída dos projetos de demonstração do FreeRTOS para reduzir o tamanho da ROM. Em vez disso, para fins de teste de laboratório, crie uma chave privada em sua estação de trabalho e use-a para assinar a CSR.

```
openssl genrsa -out tempCsrSigner.key 2048
openssl req -new -key tempCsrSigner.key -out deviceCert.csr
```

Depois que sua autoridade de certificação tiver sido criada e registrada com a AWS IoT, use o seguinte comando para emitir um certificado de cliente baseado na CSR do dispositivo assinada na etapa anterior:

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key
 -CAcreateserial -out deviceCert.pem -days 500 -sha256 -force_pubkey
 DevicePublicKey.pem
```

Mesmo que a CSR tenha sido assinada com uma chave privada temporária, o certificado emitido só pode ser usado com a chave privada do dispositivo real. O mesmo mecanismo poderá ser usado na produção se você armazenar a chave do signatário da CSR em hardware separado e configurar sua autoridade de certificação para que ela emita somente certificados para solicitações que foram assinadas por essa chave específica. Essa chave também deverá permanecer sob controle de um administrador designado.

Importação de certificado

Com o certificado emitido, a próxima etapa é importá-lo para o seu dispositivo. Também será necessário importar seu certificado de autoridade de certificação (CA), pois ele é necessário para que a primeira autenticação na AWS IoT seja bem-sucedida ao usar JITP. No arquivo `aws_clientcredential_keys.h` em seu projeto, defina a macro `keyCLIENT_CERTIFICATE_PEM` para ser o conteúdo de deviceCert.pem e a macro `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` para ser o conteúdo de `rootCA.pem`.

Autorização do dispositivo

Importe `deviceCert.pem` para o registro da AWS IoT conforme descrito em Usar seu próprio certificado. Você deve criar uma nova coisa da AWS IoT, associar o certificado PENDENTE e uma política à coisa e, depois, marcar o certificado como ATIVO. Todas essas etapas podem ser executadas manualmente no console da AWS IoT.

Quando o novo certificado de cliente estiver ATIVO e associado a uma coisa e uma política, execute a demonstração de autenticação mútua coreMQTT novamente. Desta vez, a conexão com o agente MQTT da AWS IoT será bem-sucedida.

Guias de conceitos básicos específicos da placa

> **Important**
>
> Essa página se refere ao repositório do Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Depois de concluir Primeiras etapas, consulte o guia da placa para obter instruções específicas sobre como começar a usar o FreeRTOS:

- Conceitos básicos do Kit de desenvolvimento do Cypress CYW943907AEVAL1F
- Conceitos básicos do Kit de desenvolvimento do Cypress CYW954907AEVAL1F
- Começar a usar o kit Cypress CY8CKIT-064S0S2-4343W
- Introdução ao kit de conectividade IoT Infineon XMC48 00
- Introdução ao kit inicial MW32x AWS IoT
- Começando com o kit MediaTek MT7697Hx de desenvolvimento
- Conceitos básicos do Microchip Curiosity PIC32MZ EF
- Introdução ao NuMaker Nuvoton -IoT-M487
- Introdução ao módulo de IoT NXP LPC54 018
- Conceitos básicos do Renesas Starter Kit+ para RX65N-2MB
- Introdução ao STMicroelectronics STM32L4 Discovery Kit IoT Node
- Começando com a Texas Instruments CC322 0SF- LAUNCHXL

- Conceitos básicos do simulador de dispositivo do Windows
- Introdução ao kit de IoT industrial Xilinx Avnet MicroZed

> **Note**
>
> Você não precisa concluir Primeiras etapas para os seguintes guias independentes de Conceitos básicos do FreeRTOS:
>
> - Conceitos básicos do Microchip ATECC608A Secure Element com o Windows Simulator
> - Introdução ao Espressif ESP32- DevKit C e ao ESP-WROVER-KIT
> - Conceitos básicos do Espressif ESP32-WROOM-32SE
> - Conceitos básicos do Espressif ESP32-S2
> - Conceitos básicos do Infineon OPTIGA Trust X e do XMC4800 IoT Connectivity Kit
> - Conceitos básicos do Nordic nRF52840-DK

Conceitos básicos do Kit de desenvolvimento do Cypress CYW943907AEVAL1F

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar o Kit de desenvolvimento do Cypress CYW943907AEVAL1F. Se você não tiver o Kit de desenvolvimento Cypress CYW943907AEVAL1F, acesse o AWS Partner Device Catalog para comprar um com nosso parceiro.

> **Note**
>
> Este tutorial orienta você a como configurar e executar a demonstração da autenticação mútua da coreMQTT. No momento, a porta do FreeRTOS para esta placa não oferece suporte a demonstrações de cliente e servidor de TCP.

Antes de começar, você deve configurar o AWS IoT e seu download do FreeRTOS para conectar seu dispositivo à nuvem da AWS. Para obter instruções, consulte Primeiras etapas.

Important

- Neste tópico, o caminho para o diretório de download do FreeRTOS é chamado de *`freertos`*.
- Caracteres de espaço no caminho *`freertos`* podem causar falhas na compilação. Ao clonar ou copiar o repositório, verifique se o caminho criado não contém caracteres de espaço.
- O tamanho máximo de um caminho de arquivo no Microsoft Windows é 260 caracteres. Caminhos longos de diretório de download do FreeRTOS podem causar falhas de compilação.
- O código-fonte pode conter links simbólicos, por isso se estiver usando o Windows para extrair o arquivo, talvez seja necessário:
  - Habilitar o modo Desenvolvedor ou,
  - Usar um console com privilégios de administrador.

  Dessa forma, o Windows pode criar links simbólicos adequadamente ao extrair o arquivamento. Caso contrário, os links simbólicos serão gravados como arquivos comuns que contêm os caminhos dos links simbólicos como texto ou estão vazios. Para obter mais informações, consulte a entrada no blog Symlinks no Windows 10!.

  Se você usa o Git no Windows, você deve habilitar o modo Desenvolvedor ou deve:

  - Definir `core.symlinks` como verdadeiro com o seguinte comando:

    ```
    git config --global core.symlinks true
    ```

  - Use um console com privilégios de administrador sempre que usar um comando git que grava no sistema (por exemplo, git pull, git clone, e git submodule update --init --recursive).
- Como visto em Fazer download do FreeRTOS, as portas do FreeRTOS para Cypress estão disponíveis no momento apenas no GitHub.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilar um aplicativo de demonstração do FreeRTOS de forma cruzada para uma imagem binária.
3. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.
4. Interagir com o aplicativo em execução na placa em uma conexão serial para fins de monitoramento e depuração.

Configuração do seu ambiente de desenvolvimento

Download e instalação do SDK WICED Studio

Neste Guia de conceitos básicos, use o SDK Cypress WICED Studio para programar sua placa com a demonstração do FreeRTOS. Visite o site do WICED Software para baixar o SDK do WICED Studio do Cypress. Você deve se cadastrar para uma conta gratuita do Cypress para baixar o software. O SDK do WICED Studio é compatível com os sistemas operacionais Windows, macOS e Linux.

> **Note**
>
> Alguns sistemas operacionais requerem etapas de instalações adicionais. Leia e siga todas as instruções de instalação para o sistema operacional e versão do WICED Studio que você está instalando.

Definição de variáveis de ambiente

Antes de usar o WICED Studio para programar sua placa, você deve criar uma variável de ambiente para o diretório de instalação do SDK do WICED Studio. Se o WICED Studio estiver sendo executado enquanto você cria as variáveis, você precisará reiniciar o aplicativo depois de definir as variáveis.

> **Note**
>
> O instalador do WICED Studio cria duas pastas separadas chamadas `WICED-Studio-`*`m`*`.`*`n`* em sua máquina, onde `m` e `n` são os números das versões principal e secundária, respectivamente. Este documento utiliza um nome de pasta `WICED-Studio-6.2`, mas certifique-se de usar o nome correto para a versão que você instalar. Ao definir a variável de ambiente `WICED_STUDIO_SDK_PATH`, especifique o caminho de instalação completo do SDK do WICED Studio, não o caminho de instalação do IDE do WICED Studio. No Windows e no macOS, a pasta `WICED-Studio-`*`m`*`.`*`n`* para o SDK é criada na pasta `Documents` por padrão.

Para criar a variável de ambiente no Windows

1. Abra Control Panel (Painel de controle), escolha System (Sistema), e selecione Advanced System Settings (Configurações avançadas do sistema).
2. Na aba Advanced (Avançado), selecione Environment Variables (Variáveis de ambiente).
3. Em User variables (Variáveis do usuário), escolha New (Novo).
4. Em Variable name (Nome da variável), insira **`WICED_STUDIO_SDK_PATH`**. Para Variable value (Valor da variável), insira o diretório de instalação do SDK do WICED Studio.

Para criar a variável de ambiente no Linux ou macOS

1. Abra o arquivo `/etc/profile` em seu computador e adicione a seguinte última linha do arquivo:

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. Reinicie a máquina.
3. Abra um terminal e execute os seguintes comandos:

   ```
   cd freertos/vendors/cypress/WICED_SDK
   ```

   ```
   perl platform_adjust_make.pl
   ```

   ```
   chmod +x make
   ```

Estabelecimento de uma conexão serial

Para estabelecer uma conexão em série entre sua máquina host e sua placa

1. Conecte a placa ao computador host usando um cabo USB padrão A para USB micro-B.
2. Identifique o número da porta serial USB para a conexão à placa em seu computador host.
3. Inicie um terminal serial e abra uma conexão com as seguintes configurações:
   - Taxa de baud: 115200
   - Dados: 8 bits
   - Paridade: nenhum
   - Bits de parada: 1
   - Controle de fluxo: nenhum

Para obter mais informações sobre como instalar um terminal e configurar uma conexão serial, consulte Instalação de um emulador de terminal.

Monitoramento de mensagens MQTT na nuvem

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console do AWS IoT para monitorar as mensagens enviadas pelo dispositivo para a nuvem da AWS.

Para assinar o tópico MQTT com o cliente MQTT do AWS IoT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Compilação e execução do projeto de demonstração do FreeRTOS

Depois de configurar uma conexão serial para sua placa, você poderá compilar o projeto de demonstração do FreeRTOS na placa e executar a demonstração.

Para compilar e executar o projeto de demonstração do FreeRTOS no WICED Studio

1. Inicie o WICED Studio.
2. No menu File (Arquivo), escolha Import (Importar). Expanda a pasta `General`, selecione Existing Projects into Workspace (Projetos existentes no Workspace) e escolha Next (Próximo).
3. Em Select root directory (Selecionar diretório raiz), selecione Browse... (Procurar...), navegue até o caminho *`freertos`*`/projects/cypress/CYW943907AEVAL1F/wicedstudio` e selecione OK.
4. Em Projects (Projetos), marque a caixa apenas para o projeto aws_demo. Selecione Finish (Concluir) para importar o projeto. O projeto de destino aws_demo deve aparecer na janela Make Target (Tornar destino).
5. Expanda o menu WICED Platform (Plataforma WICED) e selecione WICED Filters off (Filtros do WICED desativados).
6. Na janela Make Target (Tornar destino), expanda aws_demo, clique com o botão direito do mouse no arquivo `demo.aws_demo` e selecione Build Target (Criar destino) e faça download da demonstração para a placa. A demonstração deve ser executada automaticamente após a criação e download dela para a placa.

Solução de problemas

- Se estiver usando o Windows, você pode receber um erro a seguir quando você cria e executa o projeto demo:

  ```
  : recipe for target 'download_dct' failed
  make.exe[1]: *** [download_dct] Error 1
  ```

  Para resolver esse erro, faça o seguinte:

  1. Navegue até *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32` e clique duas vezes em `openocd-all-brcm-libftdi.exe`.
  2. Navegue até *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1` e clique duas vezes em `InstallDriver.exe`.
- Se estiver usando o Linux ou macOS, você pode receber um erro a seguir quando você cria e executa o projeto demo:

  ```
  make[1]: *** [download_dct] Error 127
  ```

Para solucionar esse erro, use o seguinte comando para atualizar o pacote libusb-dev:

```
sudo apt-get install libusb-dev
```

Para obter mais informações sobre soluções de problemas gerais sobre os Conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Conceitos básicos do Kit de desenvolvimento do Cypress CYW954907AEVAL1F

Important

Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar o Kit de desenvolvimento do Cypress CYW954907AEVAL1F. Se você não tiver o Kit de desenvolvimento Cypress CYW954907AEVAL1F, acesse o AWS Partner Device Catalog para comprar um com nosso parceiro.

Note

Este tutorial orienta você a como configurar e executar a demonstração da autenticação mútua da coreMQTT. No momento, a porta do FreeRTOS para esta placa não oferece suporte a demonstrações de cliente e servidor de TCP.

Antes de começar, você deve configurar o AWS IoT e seu download do FreeRTOS para conectar seu dispositivo à nuvem da AWS. Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de download do FreeRTOS é chamado *`freertos`*.

Important

- Neste tópico, o caminho para o diretório de download do FreeRTOS é chamado de *`freertos`*.

- Caracteres de espaço no caminho `freertos` podem causar falhas na compilação. Ao clonar ou copiar o repositório, verifique se o caminho criado não contém caracteres de espaço.
- O tamanho máximo de um caminho de arquivo no Microsoft Windows é 260 caracteres. Caminhos longos de diretório de download do FreeRTOS podem causar falhas de compilação.
- O código-fonte pode conter links simbólicos, por isso se estiver usando o Windows para extrair o arquivo, talvez seja necessário:
  - Habilitar o modo Desenvolvedor ou,
  - Usar um console com privilégios de administrador.

  Dessa forma, o Windows pode criar links simbólicos adequadamente ao extrair o arquivamento. Caso contrário, os links simbólicos serão gravados como arquivos comuns que contêm os caminhos dos links simbólicos como texto ou estão vazios. Para obter mais informações, consulte a entrada no blog Symlinks no Windows 10!.

  Se você usa o Git no Windows, você deve habilitar o modo Desenvolvedor ou deve:
  - Definir `core.symlinks` como verdadeiro com o seguinte comando:

    ```
    git config --global core.symlinks true
    ```

  - Use um console com privilégios de administrador sempre que usar um comando git que grava no sistema (por exemplo, git pull, git clone, e git submodule update --init --recursive).
- Como visto em Fazer download do FreeRTOS, as portas do FreeRTOS para Cypress estão disponíveis no momento apenas no GitHub.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilar um aplicativo de demonstração do FreeRTOS de forma cruzada para uma imagem binária.
3. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.

4. Interagir com o aplicativo em execução na placa em uma conexão serial para fins de monitoramento e depuração.

Configuração do seu ambiente de desenvolvimento

Download e instalação do SDK WICED Studio

Neste Guia de conceitos básicos, use o SDK Cypress WICED Studio para programar sua placa com a demonstração do FreeRTOS. Visite o site do WICED Software para baixar o SDK do WICED Studio do Cypress. Você deve se cadastrar para uma conta gratuita do Cypress para baixar o software. O SDK do WICED Studio é compatível com os sistemas operacionais Windows, macOS e Linux.

> **Note**
>
> Alguns sistemas operacionais requerem etapas de instalações adicionais. Leia e siga todas as instruções de instalação para o sistema operacional e versão do WICED Studio que você está instalando.

Definição de variáveis de ambiente

Antes de usar o WICED Studio para programar sua placa, você deve criar uma variável de ambiente para o diretório de instalação do SDK do WICED Studio. Se o WICED Studio estiver sendo executado enquanto você cria as variáveis, você precisará reiniciar o aplicativo depois de definir as variáveis.

> **Note**
>
> O instalador do WICED Studio cria duas pastas separadas chamadas `WICED-Studio-`*`m`*`.`*`n`* em sua máquina, onde `m` e `n` são os números das versões principal e secundária, respectivamente. Este documento utiliza um nome de pasta `WICED-Studio-6.2`, mas certifique-se de usar o nome correto para a versão que você instalar. Ao definir a variável de ambiente `WICED_STUDIO_SDK_PATH`, especifique o caminho de instalação completo do SDK do WICED Studio, não o caminho de instalação do IDE do WICED Studio. No Windows e no macOS, a pasta `WICED-Studio-`*`m`*`.`*`n`* para o SDK é criada na pasta `Documents` por padrão.

Para criar a variável de ambiente no Windows

1. Abra Control Panel (Painel de controle), escolha System (Sistema), e selecione Advanced System Settings (Configurações avançadas do sistema).
2. Na aba Advanced (Avançado), selecione Environment Variables (Variáveis de ambiente).
3. Em User variables (Variáveis do usuário), escolha New (Novo).
4. Em Variable name (Nome da variável), insira **`WICED_STUDIO_SDK_PATH`**. Para Variable value (Valor da variável), insira o diretório de instalação do SDK do WICED Studio.

Para criar a variável de ambiente no Linux ou macOS

1. Abra o arquivo `/etc/profile` em seu computador e adicione a seguinte última linha do arquivo:

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. Reinicie a máquina.
3. Abra um terminal e execute os seguintes comandos:

   ```
   cd freertos/vendors/cypress/WICED_SDK
   ```

   ```
   perl platform_adjust_make.pl
   ```

   ```
   chmod +x make
   ```

Estabelecimento de uma conexão serial

Para estabelecer uma conexão em série entre sua máquina host e sua placa

1. Conecte a placa ao computador host usando um cabo USB padrão A para USB micro-B.
2. Identifique o número da porta serial USB para a conexão à placa em seu computador host.
3. Inicie um terminal serial e abra uma conexão com as seguintes configurações:

   - Taxa de baud: 115200
   - Dados: 8 bits

- Paridade: nenhum
- Bits de parada: 1
- Controle de fluxo: nenhum

Para obter mais informações sobre como instalar um terminal e configurar uma conexão serial, consulte Instalação de um emulador de terminal.

Monitoramento de mensagens MQTT na nuvem

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console do AWS IoT para monitorar as mensagens enviadas pelo dispositivo para a nuvem da AWS.

Para assinar o tópico MQTT com o cliente MQTT do AWS IoT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Compilação e execução do projeto de demonstração do FreeRTOS

Depois de configurar uma conexão serial para sua placa, você poderá compilar o projeto de demonstração do FreeRTOS na placa e executar a demonstração.

Para compilar e executar o projeto de demonstração do FreeRTOS no WICED Studio

1. Inicie o WICED Studio.
2. No menu File (Arquivo), escolha Import (Importar). Expanda a pasta `General`, selecione Existing Projects into Workspace (Projetos existentes no Workspace) e escolha Next (Próximo).
3. Em Select root directory (Selecionar diretório raiz), selecione Browse... (Procurar...), navegue até o caminho *`freertos`*`/projects/cypress/CYW954907AEVAL1F/wicedstudio` e selecione OK.
4. Em Projects (Projetos), marque a caixa apenas para o projeto aws_demo. Selecione Finish (Concluir) para importar o projeto. O projeto de destino aws_demo deve aparecer na janela Make Target (Tornar destino).

5. Expanda o menu WICED Platform (Plataforma WICED) e selecione WICED Filters off (Filtros do WICED desativados).
6. Na janela Make Target (Tornar destino), expanda aws_demo, clique com o botão direito do mouse no arquivo `demo.aws_demo` e selecione Build Target (Criar destino) e faça download da demonstração para a placa. A demonstração deve ser executada automaticamente após a criação e download dela para a placa.

Solução de problemas

- Se estiver usando o Windows, você pode receber um erro a seguir quando você cria e executa o projeto demo:

  ```
  : recipe for target 'download_dct' failed
  make.exe[1]: *** [download_dct] Error 1
  ```

  Para resolver esse erro, faça o seguinte:

  1. Navegue até *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32` e clique duas vezes em `openocd-all-brcm-libftdi.exe`.
  2. Navegue até *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1` e clique duas vezes em `InstallDriver.exe`.
- Se estiver usando o Linux ou macOS, você pode receber um erro a seguir quando você cria e executa o projeto demo:

  ```
  make[1]: *** [download_dct] Error 127
  ```

  Para solucionar esse erro, use o seguinte comando para atualizar o pacote libusb-dev:

  ```
  sudo apt-get install libusb-dev
  ```

Para obter mais informações sobre soluções de problemas gerais sobre os Conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Começar a usar o kit Cypress CY8CKIT-064S0S2-4343W

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar o kit CY8CKIT-064S0S2-4343W. Você pode usar esse link para comprar um kit, caso ainda não tenha. Você também pode usar esse link para acessar o guia do usuário do kit.

Conceitos básicos

Antes de começar, você deve configurar o AWS IoT e o FreeRTOS para conectar seu dispositivo à nuvem da AWS. Para obter instruções, consulte Primeiras etapas. Depois de concluir os pré-requisitos, você terá um pacote do FreeRTOS com as credenciais do AWS IoT Core.

> **Note**
>
> Neste tutorial, o caminho para o diretório de download do FreeRTOS criado na seção "Primeiras etapas" é chamado *freertos*.

Configuração do ambiente de desenvolvimento

O FreeRTOS funciona com um fluxo de compilação CMake ou Make. Você pode usar o ModusToolbox para o fluxo de compilação do Make. Você pode usar o IDE do Eclipse fornecido com o ModusToolbox ou um IDE de parceiro, como IAR EW-ARM, MDK de Arm ou Microsoft Visual Studio Code. O IDE do Eclipse é compatível com os sistemas operacionais Windows, macOS e Linux.

Antes de começar, faça o download e instale o software ModusToolbox mais recente. Para obter mais informações, consulte Guia de instalação do ModusToolbox.

Ferramentas de atualização para o ModusToolbox 2.1 ou mais antigo

Se você estiver usando o IDE do Eclipse ModusToolbox 2.1 para programar este kit, precisará atualizar as ferramentas OpenOCD e Firmware-Loader.

Nas etapas a seguir, por padrão, o caminho *ModusToolbox* para:

- Do Windows é `C:\Users\`*user_name*`\ModusToolbox`.
- Do Linux é *user_home*`/ModusToolbox` ou onde você escolher extrair o arquivo.
- Do macOS está na pasta Aplicativos no volume selecionado no assistente.

Atualização do OpenOCD

Este kit exige o Cypress OpenOCD 4.0.0 ou posterior para apagar e programar o chip com êxito.

Como atualizar o Cypress OpenOCD

1. Acesse a página de lançamento do Cypress OpenOCD.
2. Faça download do arquivo para seu sistema operacional (Windows/Mac/Linux).
3. Exclua os arquivos existentes em *ModusToolbox*`/tools_2.x/openocd`.
4. Substitua os arquivos *ModusToolbox*`/tools_2.x/openocd` pelo conteúdo extraído do arquivo que você baixou na etapa anterior.

Atualização do carregador de firmware

Este kit exige o Cypress Firmware-Loader 3.0.0 ou posterior.

Como atualizar o Cypress Firmware-Loader

1. Acesse a página de lançamento do Cypress Firmware-Loader.
2. Faça download do arquivo para seu sistema operacional (Windows/Mac/Linux).
3. Exclua os arquivos existentes em *ModusToolbox*`/tools_2.x/fw-loader`.
4. Substitua os arquivos *ModusToolbox*`/tools_2.x/fw-loader` pelo conteúdo extraído do arquivo que você baixou na etapa anterior.

Como alternativa, você pode usar o CMake para gerar arquivos de compilação do projeto a partir do código-fonte do aplicativo FreeRTOS, compilar o projeto usando sua ferramenta de compilação

preferida e, em seguida, programar o kit usando o OpenOCD. Se voê preferir usar uma ferramenta de GUI para programar com o fluxo CMake, baixe e instale o Cypress Programmer a partir da página Soluções de programação da Cypress. Para obter mais informações, consulte Uso da CMake com o FreeRTOS.

Configuração do hardware

Siga estas etapas para configurar o hardware do kit.

1. Provisionamento do kit

   Siga as instruções do Guia de provisionamento do kit CY8CKIT-064S0S2-4343W para provisionar com segurança o kit para o AWS IoT.

   Este kit exige o CySecureTools 3.1.0 ou posterior.
2. Configuração de uma conexão serial

   a. Conecte o kit ao computador host.
   b. A porta serial USB do kit é enumerada automaticamente no computador host. Identifique o número da porta. No Windows, é possível identificá-lo usando o Gerenciador de dispositivos em Portas (COM e LPT).
   c. Inicie um terminal serial e abra uma conexão com as seguintes configurações:

      - Taxa de baud: 115200
      - Dados: 8 bits
      - Paridade: nenhum
      - Bits de parada: 1
      - Controle de fluxo: nenhum

Compilação e execução do projeto de demonstração do FreeRTOS

Nesta seção, você criará e executará a demonstração.

1. Siga as etapas presentes no Guia de provisionamento do kit CY8CKIT-064S0S2-4343W.
2. Compile a demonstração do FreeRTOS.

   a. Abra o IDE do Eclipse para ModusToolbox e escolha ou crie um espaço de trabalho.
   b. No menu File (Arquivo), escolha Import (Importar).

Expanda Geral, escolha Projetos existentes no espaço de trabalho e, em seguida, Próximo.

c. No Diretório raiz, insira *freertos*/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos e selecione o nome do projeto aws_demos. Isso deve ser selecionado por padrão.

d. Escolha Concluir para importar o projeto para o espaço de trabalho.

e. Compile o aplicativo fazendo uma das seguintes etapas:

- No Painel rápido, selecione Compilar aplicativo aws_demos.
- Escolha Projeto e escolha Compilar tudo.

Verifique se o projeto é compilado sem erros.

3. Monitoramento de mensagens MQTT na Nuvem

Antes de executar a demonstração, você pode configurar o cliente MQTT no console do AWS IoT para monitorar as mensagens enviadas pelo dispositivo para a nuvem da AWS. Para assinar o tópico MQTT com o cliente MQTT do AWS IoT, siga essas etapas.

a. Faça login no console do AWS IoT.

b. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.

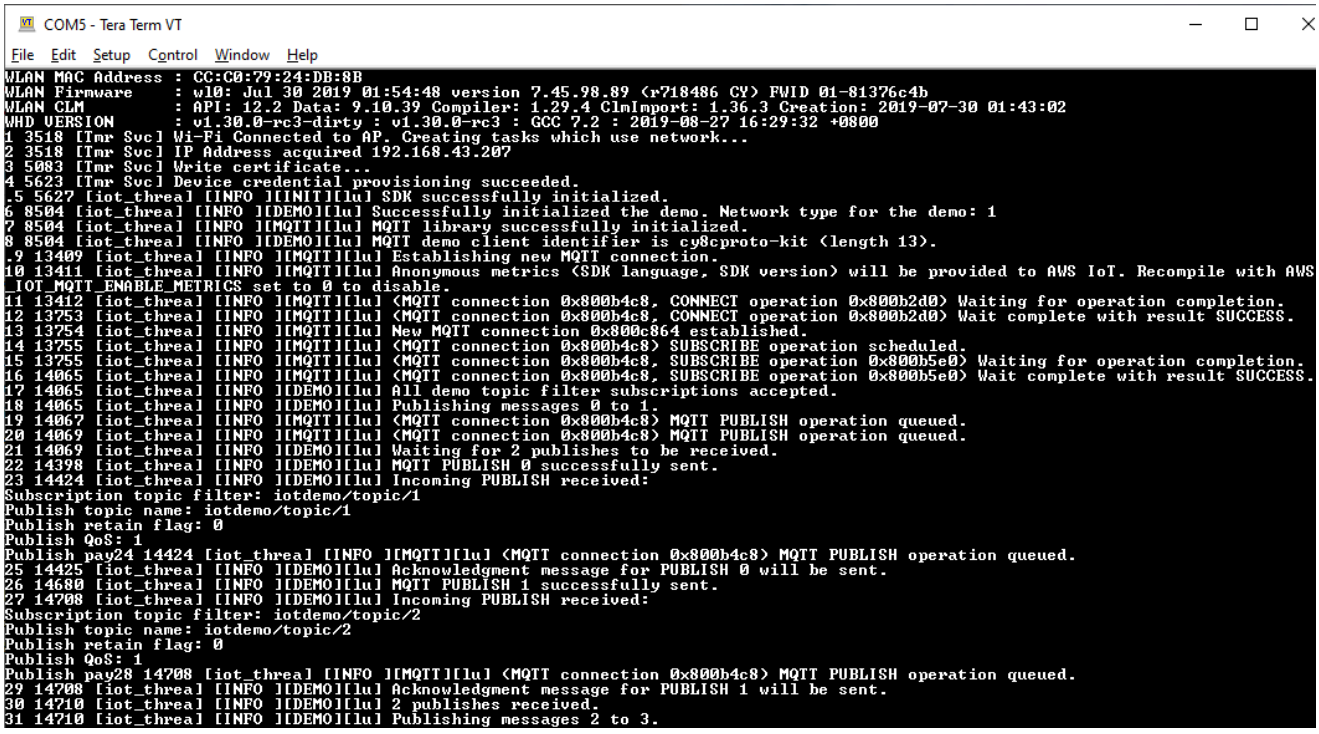
c. Em Tópico de assinatura, insira ***your-thing-name*/example/topic** e selecione Assinar o tópico.

4. Execução do projeto de demonstração do FreeRTOS

a. Selecione o projeto aws_demos no espaço de trabalho.

b. No Painel rápido, selecione Programa aws_demos (KitProg3). Isso programa a placa e o aplicativo de demonstração começa a ser executado após a conclusão da programação.

c. Você pode exibir o status da aplicação em execução no terminal serial. A figura a seguir mostra uma parte da saída do terminal.



A demonstração do MQTT publica mensagens de quatro tópicos diferentes (`iotdemo/topic/`*n*, em que n=1 a 4) e assina todos esses tópicos para receber as mesmas mensagens de volta. Quando uma mensagem é recebida, a demonstração publica uma mensagem de confirmação sobre o tópico `iotdemo/acknowledgements`. A lista a seguir descreve as mensagens de depuração que aparecem na saída do terminal, com referências aos números de série das mensagens. Na saída, os detalhes do driver WICED Host Driver (WHD) são impressos primeiro sem numeração de série.

1. De 1 a 4: o dispositivo se conecta ao ponto de acesso (AP) configurado e é provisionado conectando-se ao servidor da AWS usando o endpoint e os certificados configurados.
2. De 5 a 13: a biblioteca coreMQTT é inicializada e o dispositivo estabelece a conexão MQTT.
3. De 14 a 17: o dispositivo assina todos os tópicos para receber de volta as mensagens publicadas.
4. De 18 a 30: o dispositivo publica duas mensagens e espera para recebê-las de volta. Quando cada mensagem é recebida, o dispositivo envia uma mensagem de confirmação.

O mesmo ciclo de publicação, recebimento e confirmação continua até que todas as mensagens sejam publicadas. Duas mensagens são publicadas por ciclo até que o número de ciclos configurados seja concluído.

5. Uso da CMake com o FreeRTOS

   Você também pode usar o CMake para compilar e executar o aplicativo de demonstração. Para configurar o CMake e um sistema de compilação nativo, consulte Pré-requisitos.

   a. Use o comando a seguir para gerar os arquivos de compilação. Especifique a placa de destino com a opção `-DBOARD`.

   ```
   cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
   S freertos -B build_dir
   ```

   Se você estiver usando o Windows, deverá especificar o sistema de compilação nativo, usando a opção `-G`, porque o CMake usa o Visual Studio por padrão.

   Example

   ```
   cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
   S freertos -B build_dir -G Ninja
   ```

   Se `arm-none-eabi-gcc` não estiver no caminho do shell, você também precisará definir a variável `AFR_TOOLCHAIN_PATH` CMake.

   Example

   ```
    -DAFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
   ```

   b. Use o comando a seguir para compilar o projeto usando o CMake.

   ```
   cmake --build build_dir
   ```

   c. Por fim, programe os arquivos `cm0.hex` e `cm4.hex` gerados em *`build_dir`* usando o Cypress Programmer.

Execução de outras demonstrações

Os aplicativos de demonstração a seguir foram testados e verificados para funcionar com a versão atual. Você pode encontrar essas demonstrações no diretório *`freertos`*`/demos`. Para obter informações sobre como executar essas demonstrações, consulte Demonstrações do FreeRTOS.

- Demonstração do Bluetooth Low Energy

- Demonstração de atualizações sem fios
- Demonstração do cliente Echo de Secure Sockets
- Demonstração da solução Device Shadow do AWS IoT

Depuração

O KitProg3 do kit oferece suporte à depuração pelo protocolo SWD.

- Para depurar o aplicativo FreeRTOS , selecione o projeto aws_demos no espaço de trabalho e, em seguida, selecione Depuração aws_demos (KitProg3) no Painel rápido.

Atualizações OTA

Os MCUs do PSoC 64 foram aprovados em todos os testes de qualificação exigidos do FreeRTOS. No entanto, o atributo sem fios opcional, implementado na biblioteca de firmware da AWS PSoC 64 Standard Secure, ainda está aguardando avaliação. O atributo OTA, conforme implementado no momento, é aprovado em todos os testes de qualificação OTA, exceto em aws_ota_test_case_rollback_if_unable_to_connect_after_update.py.

Quando uma imagem OTA validada com êxito é aplicada a um dispositivo usando o PSoc64 Standard Secure: o MCU da AWS e o dispositivo não conseguem se comunicar com o AWS IoT Core, o dispositivo não pode reverter automaticamente para a imagem original em boas condições. Isso pode fazer com que o dispositivo fique inacessível do AWS IoT Core para mais atualizações. Essa funcionalidade ainda está sendo desenvolvida pela equipe do Cypress.

Para obter mais informações, consulte Atualizações OTA com a AWS e o kit CY8CKIT-064S0S2-4343W. Caso existam mais dúvidas ou você precise de suporte técnico, entre em contato com a Comunidade de desenvolvedores da Cypress.

Conceitos básicos do Microchip ATECC608A Secure Element com o Windows Simulator

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar o elemento seguro do Microchip ATECC608A com o Windows Simulator.

Você precisa do seguinte hardware:

- Placa de clique de elemento seguro do Microchip ATECC608A
- SAMD21 XPlained Pro
- Adaptador MikroBUS Xplained Pro

Antes de começar, você deve configurar AWS IoT e fazer o download dos FreeRTOS para conectar seu dispositivo à nuvem. AWS Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de download do FreeRTOS é chamado de *freertos*.

Visão geral

Este tutorial contém as seguintes etapas:

1. Conectar a placa a uma máquina host.
2. Instalar o software na máquina host para desenvolver e depurar aplicativos incorporados para o microcontrolador.
3. Fazer a compilação cruzada de uma aplicação de demonstração do FreeRTOS para uma imagem binária.
4. Carregar a imagem binária do aplicativo em na placa e executar o aplicativo.

Configuração do hardware do Microchip ATECC608A

Para poder interagir com o seu dispositivo Microchip ATECC608A, primeiro programe o SAMD21.

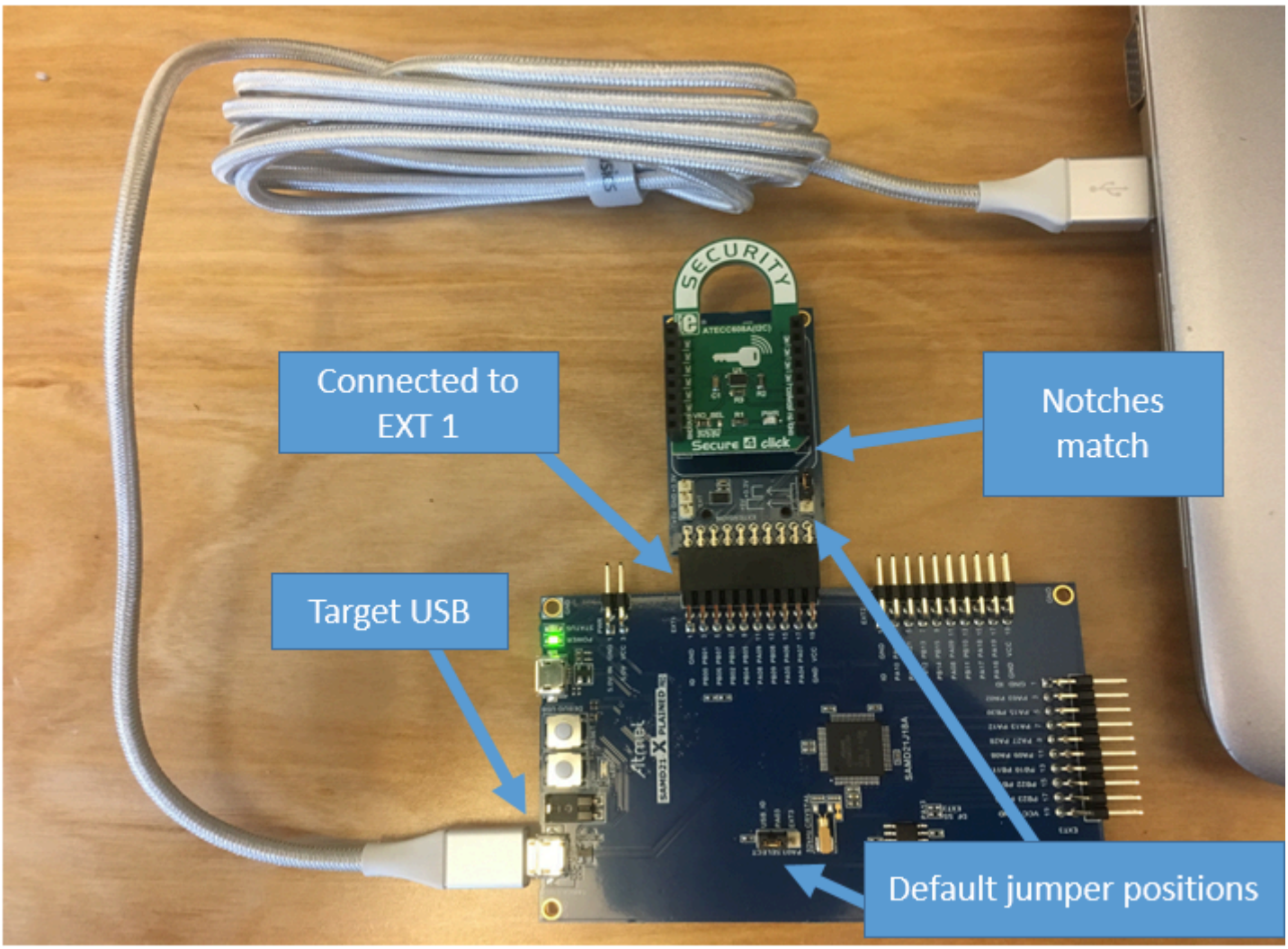
Para configurar a placa SAMD21 XPlained Pro

1. Siga o link CryptoAuthSSH-XSTK (DM320109) - Latest Firmware para baixar um arquivo.zip contendo instruções (PDF) e um binário que pode ser programado no D21.
2. Faça download e instale o IDP do Atmel Studio 7. Certifique-se de selecionar a arquitetura do driver SMART ARM MCU durante a instalação.
3. Use um cabo USB 2.0 Micro B para encaixar o conector "Debug USB" ao computador e siga as instruções no PDF. (O conector "Debug USB" é a porta USB mais próxima do LED POWER e dos pinos.)

Para conectar o hardware

1. Desconecte o cabo micro USB de Debug USB.
2. Conecte o adaptador mikroBUS XPlained Pro à placa SAMD21 no local EXT1.
3. Conecte a placa de clique do ATECC608A Secure 4 ao adaptador mikroBUSX XPlained Pro. Certifique-se de que o canto entalhado da placa de clique corresponda ao ícone entalhado na placa do adaptador.
4. Conecte o cabo micro USB a Target USB.

Sua configuração deve ser semelhante à seguinte.



Configuração do ambiente de desenvolvimento

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e inserir um código de verificação no teclado do telefone.

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

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, insira sua senha.

   Para obter ajuda ao fazer login usando o usuário raiz, consulte Fazer login como usuário raiz no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .

2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use o URL de login que foi enviado ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

Para conceder acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos em AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Criação de um conjunto de permissões no Guia do usuário do AWS IAM Identity Center .

- Usuários gerenciados no IAM com provedor de identidades:

Crie um perfil para a federação de identidades. Siga as instruções em Criando um perfil para um provedor de identidades de terceiros (federação) no Guia do Usuário do IAM.

- Usuários do IAM:
  - Crie um perfil que seu usuário possa assumir. Siga as instruções em Criação de um perfil para um usuário do IAM no Guia do usuário do IAM.
  - (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Configuração

1. Baixe o repositório FreeRTOS do repositório FreeRTOS. GitHub

   Para baixar os FreeRTOS em: GitHub

   1. Navegue até o repositório do FreeRTOS GitHub .
   2. Selecione Clone or download (Clonar ou fazer download).
   3. Na linha de comando do computador, clone o repositório para um diretório na máquina de host.

      ```
      git clone https://github.com/aws/amazon-freertos.git -\-recurse-submodules
      ```

      Important

      - Neste tópico, o caminho para o diretório de download do FreeRTOS é chamado de *freertos*.
      - Caracteres de espaço no caminho *freertos* podem causar falhas na compilação. Ao clonar ou copiar o repositório, verifique se o caminho criado não contém caracteres de espaço.
      - O tamanho máximo de um caminho de arquivo no Microsoft Windows é 260 caracteres. Caminhos longos de diretório de download do FreeRTOS podem causar falhas de compilação.
      - O código-fonte pode conter links simbólicos, por isso se estiver usando o Windows para extrair o arquivo, talvez seja necessário:
        - Habilitar o modo Desenvolvedor ou,

- Usar um console com privilégios de administrador.

Dessa forma, o Windows pode criar links simbólicos adequadamente ao extrair o arquivamento. Caso contrário, os links simbólicos serão gravados como arquivos comuns que contêm os caminhos dos links simbólicos como texto ou estão vazios. Para obter mais informações, consulte a entrada no blog Symlinks no Windows 10!.

Se você usa o Git no Windows, você deve habilitar o modo Desenvolvedor ou deve:

- Definir `core.symlinks` como verdadeiro com o seguinte comando:

```
git config -\-global core.symlinks true
```

- Use um console com privilégios de administrador sempre que usar um comando git que grava no sistema (por exemplo, git pull, git clone, e git submodule update -\-init -\-recursive).

4. No diretório *`freertos`*, confira a ramificação a ser usada.

2. Configure o ambiente de desenvolvimento.

   a. Instale a versão mais recente do WinPCap.

   b. Instale o Microsoft Visual Studio.

      As versões 2017 e 2019 do Visual Studio funcionam. Todas as edições dessas versões do Visual Studio são compatíveis (Community, Professional ou Enterprise).

      Além do IDE, instale o componente Desktop development with C++ (Desenvolvimento de desktop com C++). Em seguida, em Optional (Opcional), instale o SDK do Windows 10 mais recente.

   c. Verifique se você tem uma conexão Ethernet fixa ativa.

Compilação e execução do projeto de demonstração do FreeRTOS

Important

O dispositivo Microchip ATECC608A tem uma inicialização única que é bloqueada no dispositivo na primeira vez que um projeto é executado (durante a chamada para `C_InitToken`). No entanto, o projeto de demonstração e o projeto de teste do FreeRTOS

têm configurações diferentes. Se o dispositivo estiver bloqueado durante as configurações do projeto de demonstração, não será possível que todos os testes no projeto de teste sejam bem-sucedidos.

Para compilar e executar o projeto de demonstração do FreeRTOS com o IDE do Visual Studio

1. Carregue o projeto no Visual Studio.

   No menu File (Arquivo), escolha Open (Abrir). Escolha File/Solution (Arquivo/solução), navegue até o arquivo *freertos*`\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln` e escolha Open (Abrir).

2. Defina um novo destino para o projeto de demonstração.

   O projeto de demonstração depende do SDK do Windows, mas ele não tem uma versão do SDK do Windows especificada. Por padrão, o IDE pode tentar compilar a demonstração com uma versão do SDK que não está presente em sua máquina. Para definir a versão do SDK do Windows, clique com o botão direito do mouse em aws_demos e escolha Retarget Projects (Definir novos destinos para os projetos). Isso abre a janela Review Solution Actions (Revisar ações de solução). Escolha uma versão do SDK do Windows que esteja presente no computador (use o valor inicial na lista suspensa) e escolha OK.

3. Crie e execute o projeto.

   No menu Compilar, escolha Compilar solução, e verifique se a solução é compilada sem erros. Escolha Debug (Depurar), Start Debugging (Começar a depurar) para executar o projeto. Na primeira execução, você precisa configurar a interface do dispositivo e recompilar. Para ter mais informações, consulte Configuração da interface de rede.

4. Provisione o Microchip ATECC608A.

   A Microchip forneceu várias ferramentas de script para ajudar a configurar as peças do ATECC608A. Navegue até *freertos*`\vendors\microchip\secure_elements\app\example_trust_chain_tool` e abra o arquivo README.md.

   Siga as instruções no arquivo `README.md` para provisionar seu dispositivo. As etapas incluem o seguinte:

   1. Crie e registre uma autoridade de certificação com AWS.

2. Gerar suas chaves no Microchip ATECC608A e exportar a chave pública e o número de série do dispositivo.
3. Gere um certificado para o dispositivo e registre esse certificado com AWS.
4. Carregar o certificado CA e o certificado de dispositivo no dispositivo.
5. Criar e executar exemplos do FreeRTOS.

   Execute novamente o projeto de demonstração. Desta vez você deverá se conectar!

Solução de problemas

Para obter informações gerais sobre a solução de problemas, consulte Solução de problemas de conceitos básicos.

Introdução ao Espressif ESP32- DevKit C e ao ESP-WROVER-KIT

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

> **Note**
>
> Para explorar como integrar bibliotecas modulares e demonstrações do FreeRTOS no seu projeto Espressif IDF, consulte nossa Integração de referência em destaque para a plataforma ESP32-C3.

Siga este tutorial para começar a usar o Espressif ESP32- DevKit C equipado com os módulos ESP32-WROOM-32, ESP32-SOLO-1 ou ESP-WROVER e o ESP-WROVER-KIT-VB. Para comprar um de nosso AWS parceiro no catálogo de dispositivos parceiros, use os links a seguir:

- ESP32-QUARTO-32 C DevKit
- ESP32-SOLO-1
- ESP32-WROVER-KIT

Essas versões das placas de desenvolvimento são compatíveis com o FreeRTOS.

Para obter mais informações sobre as versões mais recentes dessas placas, consulte ESP32- DevKit C V4 ou ESP-WROVER-KIT v4.1 no site da Espressif.

> **Note**
>
> Atualmente, a porta FreeRTOS para ESP32-WROVER-KIT e DevKit ESP C não suporta o recurso de multiprocessamento simétrico (SMP).

Visão geral

Este tutorial orienta você pelas seguintes etapas:

1. Conectar sua placa a uma máquina host.
2. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
3. Compilar uma aplicação de demonstração do FreeRTOS de forma cruzada para uma imagem binária.
4. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.
5. Interagir com o aplicativo em execução na placa em uma conexão serial para fins de monitoramento e depuração.

Pré-requisitos

Antes de começar a usar os FreeRTOS em seu quadro Espressif, você deve configurar sua conta e permissões. AWS

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

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

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

Para conceder acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos em AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Criação de um conjunto de permissões no Guia do usuário do AWS IAM Identity Center .
- Usuários gerenciados no IAM com provedor de identidades:

  Crie um perfil para a federação de identidades. Siga as instruções em Criar um perfil para um provedor de identidades de terceiros (federação) no Guia do usuário do IAM.
- Usuários do IAM:
  - Crie um perfil que seu usuário possa assumir. Siga as instruções em Criação de um perfil para um usuário do IAM no Guia do usuário do IAM.

- (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Conceitos básicos

> **Note**
>
> Os comandos do Linux neste tutorial exigem o uso do shell Bash.

1. Configuração de hardware da Espressif.

   - Para obter informações sobre como configurar o hardware da placa de desenvolvimento ESP32- DevKit C, consulte o Guia de introdução do ESP32- DevKit C V4.
   - Para obter informações sobre como configurar o hardware da placa de desenvolvimento ESP-WROVER-KIT, consulte o Guia de conceitos básicos do ESP-WROVER-KIT V4.1.

   > **Important**
   >
   > Ao chegar na seção Começar a usar dos guias do Espressif, pare e retorne para as instruções desta página.

2. Faça o download dos Amazon GitHubFreeRTOS em. (Para obter instruções, consulte o arquivo README.md.)
3. Configure o ambiente de desenvolvimento.

   Para se comunicar com sua placa, você deve instalar uma cadeia de ferramentas. A Espressif fornece o ESP-IDF para desenvolver software para suas placas. Como o ESP-IDF tem a própria versão do kernel do FreeRTOS integrada como um componente, o Amazon FreeRTOS inclui uma versão personalizada do ESP-IDF v4.2 que tem o kernel do FreeRTOS removido. Isso corrige problemas com arquivos duplicados durante a compilação. Para usar a versão personalizada do ESP-IDF v4.2 incluída no Amazon FreeRTOS, siga as instruções abaixo para o sistema operacional da sua máquina host.

   Windows

1. Faça download do Instalador online universal do ESP-IDF para Windows.
2. Execute o Instalador online universal.
3. Ao chegar à etapa Fazer download ou usar ESP-IDF, selecione Usar um diretório ESP-IDF existente e defina Escolher diretório ESP-IDF existente como ***freertos*/vendors/espressif/esp-idf**.
4. Concluir a instalação.

macOS

1. Siga as instruções nos Pré-requisitos de configuração padrão da cadeia de ferramentas (ESP-IDF v4.2) para macOS.

   > **Important**
   >
   > Quando você chegar nas instruções de "Obter ESP-IDF" em Próximas etapas, pare e retorne para as instruções desta página.

2. Abra a janela de linha de comando.
3. Navegue até o diretório de download do FreeRTOS e, em seguida, execute o script a seguir para baixar e instalar a cadeia de ferramentas do espressif em sua plataforma.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Adicione as ferramentas da cadeia de ferramentas ESP-IDF ao caminho do seu terminal com o comando a seguir.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

Linux

1. Siga as instruções nos Pré-requisitos da configuração padrão da cadeia de ferramentas (ESP-IDF v4.2) para Linux.

> **Important**
>
> Quando você chegar nas instruções de "Obter ESP-IDF" em Próximas etapas, pare e retorne para as instruções desta página.

2. Abra a janela de linha de comando.
3. Navegue até o diretório de download do FreeRTOS e execute o script a seguir para baixar e instalar a cadeia de ferramentas do Espressif em sua plataforma.

```
vendors/espressif/esp-idf/install.sh
```

4. Adicione as ferramentas da cadeia de ferramentas ESP-IDF ao caminho do seu terminal com o comando a seguir.

```
source vendors/espressif/esp-idf/export.sh
```

4. Estabelecimento de uma conexão serial.

   a. Para estabelecer uma conexão serial entre sua máquina host e o ESP32- DevKit C, você deve instalar os drivers CP210x USB para UART Bridge VCP. Você pode fazer download desses drivers no Silicon Labs.

      Para estabelecer uma conexão serial entre sua máquina host e o ESP32-WROVER-KIT, é necessário instalar o driver de porta COM virtual FTDI. Você pode fazer download desse driver no FTDI.

   b. Siga as etapas para Estabelecer conexão serial com ESP32.

   c. Depois de estabelecer uma conexão serial, anote a porta serial da conexão de sua placa. Você precisa disso para instalar a demonstração.

Configuração das aplicações de demonstração do FreeRTOS

Para este tutorial, o arquivo de configuração do FreeRTOS está localizado em *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h. (Por exemplo, se AFR_BOARD espressif.esp32_devkitc for escolhido, o arquivo de configuração estará localizado em *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

1. Se você estiver executando macOS ou Linux, abra um prompt de terminal. Se você estiver executando o Windows, abra a aplicação "ESP-IDF 4.x CMD" (se você incluiu essa opção ao instalar a cadeia de ferramentas ESP-IDF) ou, caso contrário, a aplicação "Prompt de comando".
2. Para verificar se você tem o Python3 instalado, execute

   ```
   python --version
   ```

   A versão instalada é exibida. Se você não tiver o Python 3.0.1 ou posterior instalado, poderá instalá-lo a partir do site do Python.
3. Você precisa da interface de linha de AWS comando (CLI) para executar AWS IoT comandos. Se você estiver executando o Windows, use o `easy_install awscli` comando para instalar a AWS CLI no aplicativo “Command” ou “ESP-IDF 4.x CMD”.

   Se você estiver executando o macOS ou o Linux, consulte Instalação da CLI AWS.
4. Executar

   ```
   aws configure
   ```

   e configure a AWS CLI com seu ID de chave de AWS acesso, chave de acesso secreta e região padrão AWS . Para obter mais informações, consulte Configurar a CLI AWS.
5. Use o comando a seguir para instalar o AWS SDK para Python (boto3):

   - No Windows, na aplicação "Comando" ou "ESP-IDF 4.x CMD", execute

     ```
     pip install boto3 --user
     ```

     > **Note**
     >
     > Consulte os detalhes na documentação do boto3.

   - No macOS ou Linux, execute

     ```
     pip install tornado nose --user
     ```

     e depois execute

```
pip install boto3 --user
```

O FreeRTOS inclui o script `SetupAWS.py` para facilitar a configuração da placa Espressif para conectar-se ao AWS IoT. Para configurar o script, abra *`freertos`*`/tools/aws_config_quick_start/configure.json` e defina os seguintes atributos:

`afr_source_dir`

O caminho completo para o diretório *`freertos`* no computador. Certifique-se de usar barras para especificar esse caminho.

`thing_name`

O nome que você deseja atribuir à AWS IoT coisa que representa seu quadro.

`wifi_ssid`

O SSID da rede Wi-Fi.

`wifi_password`

A senha da rede Wi-Fi.

`wifi_security`

O tipo de segurança da rede Wi-Fi.

Os tipos de segurança válidos estão a seguir:

- `eWiFiSecurityOpen` (Aberto, sem segurança)
- `eWiFiSecurityWEP` (segurança WEP)
- `eWiFiSecurityWPA` (segurança WPA)
- `eWiFiSecurityWPA2` (segurança WPA2)

6. Executar o script de configuração.

   a. Se você estiver executando macOS ou Linux, abra um prompt de terminal. Se você estiver executando o Windows, abra a aplicação "ESP-IDF 4.x CMD" ou "Comando".

   b. Navegue até o diretório *`freertos`*`/tools/aws_config_quick_start` e execute

```
python SetupAWS.py setup
```

O script faz o seguinte:

- Cria uma coisa, um certificado e uma política de IoT.
- Anexa a política de IoT ao certificado e o certificado à coisa do AWS IoT .
- Preenche o arquivo `aws_clientcredential.h` com o endpoint, o SSID Wi-Fi e as credenciais da AWS IoT .
- Formata o certificado e a chave privada e os grava no arquivo de cabeçalho `aws_clientcredential_keys.h`.

> **Note**
>
> O certificado é codificado apenas para fins de demonstração. Por este motivo, as aplicações devem armazenar esses arquivos em um local seguro.

Para obter mais informações sobre `SetupAWS.py`, consulte `README.md` no diretório *`freertos`*`/tools/aws_config_quick_start`.

Monitoramento de mensagens MQTT na nuvem

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console para monitorar AWS IoT as mensagens que seu dispositivo envia para a nuvem. AWS

Para assinar o tópico MQTT com o cliente AWS IoT MQTT

1. Navegue até o console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha Cliente de teste MQTT.
3. Em Tópico de inscrição, insira *`your-thing-name`*`/example/topic` e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Compilação, atualização e execução do projeto de demonstração do FreeRTOS usando o script idf.py

Você pode usar o utilitário IDF (`idf.py`) do Espressif para criar o projeto e instalar os binários em seu dispositivo.

> **Note**
>
> Algumas configurações podem exigir que você use a opção de porta `"-p port-name"` com`idf.py` para especificar a porta correta, como no exemplo a seguir.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Compilação e instalação do FreeRTOS no Windows, Linux e macOS (ESP-IDF v4.2)

1. Navegue até a raiz do diretório de downloads do FreeRTOS.
2. Na janela da linha de comando, insira o comando a seguir para adicionar as ferramentas de ESP-IDF ao PATH do seu terminal.

   Windows (aplicação "Comando")

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (aplicação "ESP-IDF 4.x CMD")

   (Isso já foi feito quando você abriu a aplicação.)

   Linux / macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Configure o cmake no diretório `build` e compile a imagem do firmware com o comando a seguir.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 build
   ```

   O resultado deverá ser parecido com o que segue.

   ```
   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'
```

Se não houver erros, a compilação gerará os arquivos .bin binários do firmware.

4. Apague a memória instalada da placa de desenvolvimento com o comando a seguir.

```
idf.py erase_flash
```

5. Use o script `idf.py` para instalar o binário da aplicação na placa.

```
idf.py flash
```

6. Monitore a saída da porta serial da placa com o comando a seguir.

```
idf.py monitor
```

> **Note**
>
> Você pode combinar esses comandos, como no exemplo a seguir.
>
> ```
> idf.py erase_flash flash monitor
> ```

Para determinadas configurações da máquina host, você deve especificar a porta ao instalar a placa, como no exemplo a seguir.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

Compilação e instalação do FreeRTOS com o CMake

Além de usar o script `idf.py` fornecido pelo SDK do IDF para criar e executar seu código, você também pode compilar o projeto com o CMake. Atualmente, ele é compatível com o Makefile da Unix ou sistema de compilação Ninja.

Como compilar e instalar o projeto

1. Em uma janela da linha de comando, navegue até o diretório raiz de downloads do FreeRTOS.
2. Execute o script a seguir para adicionar as ferramentas ESP-IDF ao PATH do shell.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux / macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Insira o comando a seguir para gerar os arquivos de compilação.

   Com Makefiles da Unix

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
   B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
   ```

   Com 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. Crie o projeto.

Com Makefiles da Unix

```
make -C ./YOUR_BUILD_DIRECTORY -j8
```

Com Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. Apague a instalada e depois instale a placa.

   Com Makefiles da Unix

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Com Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Execução das demonstrações de Bluetooth Low-Energy

O FreeRTOS oferece suporte à conectividade Biblioteca de Bluetooth Low Energy.

Para executar o projeto de demonstração do FreeRTOS no Bluetooth Low Energy, você precisa executar a aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS em um dispositivo móvel Android ou iOS.

Como configurar a aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS

1. Siga as instruções em SDKs móveis para dispositivos Bluetooth do FreeRTOS para fazer download e instalar o SDK para sua plataforma móvel em seu computador host.
2. Siga as instruções em aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS para configurar a aplicação móvel de demonstração no dispositivo móvel.

Para obter instruções sobre como executar a demonstração do MQTT por Bluetooth Low Energy na sua placa, consulte MQTT por Bluetooth Low Energy.

Para obter instruções sobre como executar a demonstração do provisionamento de Wi-Fi na sua placa, consulte Provisionamento de Wi-Fi.

Uso do FreeRTOS em seu próprio projeto CMake para ESP32

Se desejar usar o FreeRTOS em seu próprio projeto CMake, você pode configurá-lo como um subdiretório e compilá-lo junto com sua aplicação. Primeiro, obtenha uma cópia dos FreeRTOS em. GitHub Você também pode configurá-lo como um submódulo Git com o comando a seguir para que seja mais fácil atualizar no futuro.

```
git submodule add -b release https://github.com/aws/amazon-freertos.git freertos
```

Se uma versão mais recente for lançada, você poderá atualizar sua cópia local com esses 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"
```

Se o seu projeto tem a seguinte estrutura de diretórios:

```
- freertos (the copy that you obtained from GitHub or the AWS IoT console)
- src
  - main.c (your application code)
- CMakeLists.txt
```

A seguir há um exemplo do arquivo `CMakeLists.txt` de nível superior que pode ser usado para compilar sua aplicação junto com o 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 criar o projeto, execute os seguintes comandos CMake. Certifique-se de que o compilador ESP32 está na variável de ambiente PATH.

```
cmake -S . -B build-directory -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/
xtensa-esp32.cmake -GNinja
```

```
cmake --build build-directory
```

Para instalar a aplicação na sua placa, execute o comando a seguir.

```
cmake --build build-directory --target flash
```

Uso de componentes do FreeRTOS

Depois de executar o CMake, você pode encontrar todos os componentes disponíveis na saída de resumo. Isso deve ser parecido com a seguinte exemplo.

```
====================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:
=============================================================================
```

Você pode fazer referência a qualquer componente da lista `Modules to build`. Para vinculá-los à sua aplicação, coloque o namespace `AFR::` na frente do nome, por exemplo, `AFR::core_mqtt`, `AFR::ota` e assim por diante.

Adição de componentes personalizados usando o ESP-IDF

Você pode adicionar mais componentes ao usar o ESP-IDF. Por exemplo, supondo que você deseja adicionar um componente chamado `example_component` e seu projeto tenha a seguinte aparência:

```
- freertos
- components
  - example_component
    - include
      - example_component.h
    - src
      - example_component.c
    - CMakeLists.txt
- src
  - main.c
- CMakeLists.txt
```

Veja a seguir um exemplo do arquivo `CMakeLists.txt` do seu componente.

```
add_library(example_component src/example_component.c)
target_include_directories(example_component PUBLIC include)
```

Em seguida, no arquivo `CMakeLists.txt` de nível superior, adicione o componente inserindo a linha a seguir logo depois de `add_subdirectory(freertos)`.

```
add_subdirectory(component/example_component)
```

Em seguida, modifique `target_link_libraries` para incluir seu componente.

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)
```

Esse componente agora é vinculado automaticamente ao código da aplicação por padrão. Você deve poder incluir seus arquivos de cabeçalho e chamar as funções que eles definem.

Substituição de configurações do FreeRTOS

Atualmente, não há uma abordagem bem definida para redefinir as configurações fora da árvore de origem do FreeRTOS. Por padrão, o CMake procurará os diretórios *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ e *freertos*/demos/include/. No entanto, você pode usar uma solução alternativa para dizer ao compilador para procurar outros diretórios primeiro. Por exemplo, você pode adicionar outra pasta para configurações do FreeRTOS.

```
- freertos
- freertos-configs
  - aws_clientcredential.h
  - aws_clientcredential_keys.h
  - iot_mqtt_agent_config.h
  - iot_config.h
- components
- src
- CMakeLists.txt
```

Os arquivos em `freertos-configs` são copiados dos diretórios *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ e *freertos*/demos/include/. No seu arquivo `CMakeLists.txt` de nível superior, adicione esta linha antes de `add_subdirectory(freertos)` para que o compilador pesquise este diretório primeiro.

```
include_directories(BEFORE freertos-configs)
```

Fornecimento de seu próprio sdkconfig para ESP-IDF

No caso de você querer fornecer o seu próprio `sdkconfig.default`, você pode definir a variável CMake `IDF_SDKCONFIG_DEFAULTS` na linha de comando:

```
cmake -S . -B build-directory -DIDF_SDKCONFIG_DEFAULTS=path_to_your_sdkconfig_defaults
 -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/xtensa-esp32.cmake -GNinja
```

Se não for especificado um local para seu próprio arquivo `sdkconfig.default`, o FreeRTOS usará o arquivo padrão localizado em *`freertos`*`/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults`.

Para obter mais informações, consulte Configuração do projeto na Referência da API do Espressif e, se você encontrar problemas após compilar com êxito, consulte a seção sobre Opções obsoletas e suas substituições nessa página.

Resumo

Se você tem um projeto com um componente chamado `example_component` e desejar substituir algumas configurações, aqui está um exemplo completo do arquivo `CMakeLists.txt` de nível 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)
```

Solução de problemas

- Se você estiver executando o macOS e o sistema operacional não reconhece o ESP-WROVER-KIT, verifique se os drivers D2XX não estão instalados. Para desinstalá-los, siga as instruções em FTDI Drivers Installation Guide for macOS X.
- O utilitário de monitor fornecido pelo ESP-IDF (e invocado usando o monitor make) ajuda a decodificar endereços. Por esse motivo, ele pode ajudar você a obter alguns backtraces significativos caso a aplicação falhe. Para obter mais informações, consulte Decodificação de endereços automática no site da Espressif.
- Também é possível habilitar GDBstub para comunicação com gdb sem a necessidade de nenhum hardware JTAG especial. Para obter mais informações, consulte Inicialização do GDB para GDBStub no site da Espressif.
- Para obter informações sobre a configuração de um ambiente com base em OpenOCD se a depuração com base em hardware JTAG for necessária, consulte Depuração JTAG no site da Espressif.
- Se não for possível instalar `pyserial` usando `pip` no macOS, faça download no site do pyserial.
- Se a placa for reiniciada continuamente, tente apagar a instalação digitando o seguinte comando no terminal.

```
make erase_flash
```

- Se você vir erros ao executar `idf_monitor.py`, use Python 2.7.
- As bibliotecas necessárias de ESP-IDF estão incluídas no FreeRTOS, portanto, não é necessário baixá-las externamente. Se a variável de ambiente `IDF_PATH` estiver definida, recomendamos que faça a limpeza dela antes de compilar o FreeRTOS.
- No Windows, pode levar de 3 a 4 minutos para o projeto ser criado. Para reduzir o tempo de compilação, você pode usar o comutador `-j4` no comando make.

```
make flash monitor -j4
```

- Se o dispositivo tiver problemas para se conectar AWS IoT, abra o `aws_clientcredential.h` arquivo e verifique se as variáveis de configuração estão definidas corretamente no arquivo. `clientcredentialMQTT_BROKER_ENDPOINT[]`deveria ser assim`1234567890123-ats.iot.us-east-1.amazonaws.com`.
- Se você estiver seguindo as etapas em Uso do FreeRTOS em seu próprio projeto CMake para ESP32 e vir erros de referência indefinidos no vinculador, em geral, a causa é a falta de bibliotecas

dependentes ou demonstrações. Para adicioná-los, atualize o arquivo `CMakeLists.txt` (no diretório raiz) usando a função padrão CMake `target_link_libraries`.

- O ESP-IDF v4.2 é compatível com o uso da cadeia de ferramentas xtensa\ -esp32\ -elf\ -gcc 8\ .2\ .0\. Se você estiver usando uma versão anterior da cadeia de ferramentas Xtensa, baixe a versão necessária.
- Se você ver um log de erros como o seguinte sobre dependências de python que não estão sendo atendidas no ESP-IDF v4.2:

```
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 as dependências do Python na sua plataforma usando o seguinte comando do Python:

```
root/vendors/espressif/esp-idf/requirements.txt
```

Para obter mais informações sobre solução de problemas, consulte Solução de problemas de conceitos básicos.

Depuração

Código de depuração no Espressif ESP32- DevKit C e no ESP-WROVER-KIT (ESP-IDF v4.2)

Esta seção mostra como depurar o hardware da Espressif usando o ESP-IDF v4.2. Você precisa de um cabo JTAG para USB. Usamos um cabo USB para MPSSE (por exemplo, o FTDI C232HM-DDHSL-0).

Configuração do ESP- DevKit C JTAG

Para o cabo FTDI C232HM-DDHSL-0, estas são as conexões com o 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             |
```

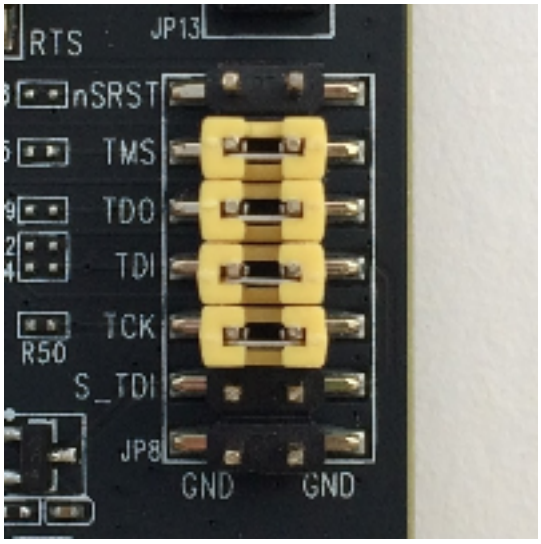
Configuração de ESP-WROVER-KIT JTAG

Para o cabo FTDI C232HM-DDHSL-0, estas são as conexões com o 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             |
```

Essas tabelas foram desenvolvidas na planilha FTDI C232HM-DDHSL-0. Para obter mais informações, consulte "Detalhes mecânicos e conexão de cabos C232HM MPSSE" na planilha.

Para habilitar a JTAG no ESP-WROVER-KIT, coloque jumpers nos pinos TMS, TDO, TDI, TCK e S_TDI, conforme mostrado aqui.



Depuração no Windows (ESP-IDF v4.2)

Para configurar para depuração no Windows

1. Conecte o lado USB do FTDI C232HM-DDHSL-0 ao computador e o outro lado como descrito em Código de depuração no Espressif ESP32- DevKit C e no ESP-WROVER-KIT (ESP-IDF v4.2). O dispositivo FTDI C232HM-DDHSL-0 deve aparecer no Device Manager (Gerenciador de dispositivos) em Universal Serial Bus Controllers (Controladores USB).
2. Na lista de dispositivos USB, clique com o botão direito do mouse no dispositivo C232HM-DDHSL-0 dispositivo e selecione Propriedades.

   > **Note**
   >
   > O dispositivo pode estar listado como USB Serial Port (Porta serial USB).

   Para ver as propriedades do dispositivo, na janela de propriedades, escolha a guia Detalhes. Se o dispositivo não estiver listado, instale o driver do Windows para FTDI C232HM-DDHSL-0.
3. Na guia Details (Detalhes), selecione Property (Propriedade) e selecione Hardware IDs (IDs de hardware). Você deve ver algo parecido com isto no campo Valor.

   ```
   FTDIBUS\COMPORT&VID_0403&PID_6014
   ```

   Neste exemplo, o ID do fornecedor é 0403 e o ID do produto é 6014.

   Verifique se esses IDs correspondem os IDs em `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Os IDs são especificados em uma linha que começa com `ftdi_vid_pid` seguidos por um ID de fornecedor e um ID do produto.

   ```
   ftdi_vid_pid 0x0403 0x6014
   ```
4. Faça download do OpenOCD para Windows.
5. Descompacte o arquivo `C:\` e adicione `C:\openocd-esp32\bin` ao caminho do sistema.
6. O OpenOCD requer libusb, que não é instalado por padrão no Windows. Para instalar o libusb:

   a. Faça download de zadig.exe.

b. Executar `zadig.exe`. No menu Options (Opções), escolha List All Devices (Listar todos os dispositivos).

c. No menu suspenso, escolha C232HM-DDHSL-0.

d. No campo de driver de destino, à direita da seta verde, selecione WinUSB.

e. Na caixa vertical no campo de driver de destino, escolha a seta e selecione Instalar driver. Escolha Replace Driver (Substituir driver).

7. Abra outro terminal, navegue até a raiz do diretório de download do FreeRTOS e execute o seguinte comando.

```
idf.py openocd
```

Deixe o prompt de comando aberto.

8. Abra outro terminal, navegue até a raiz do diretório de download do FreeRTOS e execute

```
idf.py flash monitor
```

9. Abra outro prompt de comando, navegue até a raiz do diretório de download do FreeRTOS e espere até que a demonstração comece a ser executada na placa. Quando isso ocorrer, execute

```
idf.py gdb
```

O programa deve parar na função `main`.

> **Note**
>
> O ESP32 oferece suporte a, no máximo, dois pontos de interrupção.

Depuração no macOS (ESP-IDF v4.2)

1. Faça download do driver FTDI para macOS.
2. Faça download do OpenOCD.
3. Extraia o arquivo .tar baixado e defina o caminho em `.bash_profile` como `OCD_INSTALL_DIR/openocd-esp32/bin`.
4. Use o comando a seguir para instalar o `libusb` no macOS.

```
brew install libusb
```

5. Use o comando a seguir para fazer download do driver da porta serial.

```
sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
```

6. Use o comando a seguir para fazer download do driver da porta serial.

```
sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
```

7. Se você estiver executando uma versão posterior a 10.9 do macOS, use o comando a seguir para descarregar o driver FTDI da Apple.

```
sudo kextunload -b com.apple.driver.AppleUSBFTDI
```

8. Use o seguinte comando para obter o ID do produto e o ID do fornecedor do cabo FTDI. Ele lista os dispositivos USB conectados.

```
system_profiler SPUSBDataType
```

   A saída de `system_profiler` deve ser a seguinte.

```
   DEVICE:

   Product ID: product-ID
   Vendor ID: vendor-ID (Future Technology Devices International Limited)
```

9. Abra o arquivo `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Os IDs do fornecedor e do produto do seu dispositivo são especificados em uma linha que começa com `ftdi_vid_pid`. Altere os IDs de acordo com os IDs da saída `system_profiler` na etapa anterior.
10. Abra uma janela do terminal, navegue até a raiz do diretório de download do FreeRTOS e use o seguinte comando para executar o OpenOCD.

```
idf.py openocd
```

    Deixe essa janela do terminal aberta.

11. Abra um novo terminal e use o seguinte comando para carregar o driver de porta serial FTDI.

```
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

12. Navegue até a raiz do diretório de downloads do FreeRTOS e execute

```
idf.py flash monitor
```

13. Abra outro terminal, navegue até a raiz do diretório de download do FreeRTOS e execute

```
idf.py gdb
```

O programa deve parar em `main`.

Depuração no Linux (ESP-IDF v4.2)

1. Faça download do OpenOCD. Extraia o tarball e siga as instruções de instalação no arquivo readme.
2. Use o seguinte comando para instalar libusb em Linux.

```
sudo apt-get install libusb-1.0
```

3. Abra um terminal e digite ls -l /dev/ttyUSB* para listar todos os dispositivos USB conectados ao computador. Isso ajuda você a verificar se as portas USB da placa são reconhecidas pelo sistema operacional. O resultado deverá ser parecido com o que segue.

```
   $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. Saia e, em seguida, faça login e desligue e ligue a alimentação da placa para que as alterações entrem em vigor. Em um prompt de terminal, liste os dispositivos USB. Certifique-se de que o proprietário do grupo tenha sido alterado de `dialout` para `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
```

A interface `/dev/ttyUSBn` com o menor número é usada para comunicação JTAG. A outra interface é roteada para a porta serial do ESP32 (UART) e é usada para fazer upload de código na memória flash do ESP32.

5. Na janela do terminal, navegue até a raiz do diretório de download do FreeRTOS e use o seguinte comando para executar o OpenOCD.

```
idf.py openocd
```

6. Abra outro terminal, navegue até a raiz do diretório de download do FreeRTOS e execute o seguinte comando.

```
idf.py flash monitor
```

7. Abra outro terminal, navegue até a raiz do diretório de download do FreeRTOS e execute o seguinte comando.

```
idf.py gdb
```

O programa deve parar em `main()`.

Conceitos básicos do Espressif ESP32-WROOM-32SE

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

> **Note**
>
> - Para explorar como integrar bibliotecas modulares e demonstrações do FreeRTOS no seu projeto Espressif IDF, consulte nossa Integração de referência em destaque para a plataforma ESP32-C3.
> - Atualmente, a porta FreeRTOS para ESP32-WROOM-32SE não é compatível com o recurso de multiprocessamento simétrico (SMP).

Este tutorial mostra como começar a usar o Espressif ESP32-WROOM-32SE. Para comprar um de nosso parceiro no catálogo de dispositivos AWS parceiros, consulte ESP32-WROOM-32SE.

Visão geral

Este tutorial orienta você pelas seguintes etapas:

1. Conectar a placa a uma máquina host.
2. Instalar o software na máquina host para desenvolver e depurar aplicações incorporadas para a placa do microcontrolador.
3. Compilar uma aplicação de demonstração do FreeRTOS de forma cruzada para uma imagem binária.
4. Carregar a imagem binária do aplicativo em na placa e executar o aplicativo.
5. Monitorar e depurar a aplicação em execução usando uma conexão serial.

Pré-requisitos

Antes de começar a usar os FreeRTOS em seu quadro Espressif, você deve configurar sua conta e permissões. AWS

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

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

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

Para conceder acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos em AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Criação de um conjunto de permissões no Guia do usuário do AWS IAM Identity Center .
- Usuários gerenciados no IAM com provedor de identidades:

  Crie um perfil para a federação de identidades. Siga as instruções em Criar um perfil para um provedor de identidades de terceiros (federação) no Guia do usuário do IAM.
- Usuários do IAM:
  - Crie um perfil que seu usuário possa assumir. Siga as instruções em Criação de um perfil para um usuário do IAM no Guia do usuário do IAM.

- (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Conceitos básicos

> **Note**
>
> Os comandos do Linux neste tutorial exigem o uso do shell Bash.

1. Configuração de hardware da Espressif.

   Para obter informações sobre como configurar o hardware da placa de desenvolvimento ESP32-WROOM-32SE, consulte o Guia de introdução do ESP32-C V4. DevKit

   > **Important**
   >
   > Ao chegar à seção Guia detalhado de instalação deste guia, siga até concluir a Etapa 4 (Configurar as variáveis de ambiente). Pare depois de concluir a Etapa 4 e siga as etapas restantes aqui.

2. Faça o download dos Amazon GitHubFreeRTOS em. (Para obter instruções, consulte o arquivo README.md.)
3. Configure o ambiente de desenvolvimento.

   Para se comunicar com sua placa, você deve instalar uma cadeia de ferramentas. A Espressif fornece o ESP-IDF para desenvolver software para suas placas. Como o ESP-IDF tem a própria versão do kernel do FreeRTOS integrada como um componente, o Amazon FreeRTOS inclui uma versão personalizada do ESP-IDF v4.2 que tem o kernel do FreeRTOS removido. Isso corrige problemas com arquivos duplicados durante a compilação. Para usar a versão personalizada do ESP-IDF v4.2 incluída no Amazon FreeRTOS, siga as instruções abaixo para o sistema operacional da sua máquina host.

   Windows

   1. Faça download do Instalador online universal do ESP-IDF para Windows.
   2. Execute o Instalador online universal.

3. Ao chegar à etapa Fazer download ou usar ESP-IDF, selecione Usar um diretório ESP-IDF existente e defina Escolher diretório ESP-IDF existente como ***freertos*****/vendors/espressif/esp-idf**.
4. Concluir a instalação.

macOS

1. Siga as instruções nos Pré-requisitos de configuração padrão da cadeia de ferramentas (ESP-IDF v4.2) para macOS.

> **Important**
>
> Quando você chegar nas instruções de "Obter ESP-IDF" em Próximas etapas, pare e retorne para as instruções desta página.

2. Abra a janela de linha de comando.
3. Navegue até o diretório de download do FreeRTOS e, em seguida, execute o script a seguir para baixar e instalar a cadeia de ferramentas do espressif em sua plataforma.

```
vendors/espressif/esp-idf/install.sh
```

4. Adicione as ferramentas da cadeia de ferramentas ESP-IDF ao caminho do seu terminal com o comando a seguir.

```
source vendors/espressif/esp-idf/export.sh
```

Linux

1. Siga as instruções nos Pré-requisitos da configuração padrão da cadeia de ferramentas (ESP-IDF v4.2) para Linux.

> **Important**
>
> Quando você chegar nas instruções de "Obter ESP-IDF" em Próximas etapas, pare e retorne para as instruções desta página.

2. Abra a janela de linha de comando.

3. Navegue até o diretório de download do FreeRTOS e execute o script a seguir para baixar e instalar a cadeia de ferramentas do Espressif em sua plataforma.

```
vendors/espressif/esp-idf/install.sh
```

4. Adicione as ferramentas da cadeia de ferramentas ESP-IDF ao caminho do seu terminal com o comando a seguir.

```
source vendors/espressif/esp-idf/export.sh
```

4. Estabelecimento de uma conexão serial.

   a. Para estabelecer uma conexão serial entre a máquina host e o ESP32-WROOM-32SE, instale CP210x USB nos drivers de UART Bridge VCP. Você pode fazer download desses drivers no Silicon Labs.

   b. Siga as etapas para Estabelecer uma conexão serial com ESP32.

   c. Depois de estabelecer uma conexão serial, anote a porta serial da conexão de sua placa. Você precisa disso para instalar a demonstração.

Configuração das aplicações de demonstração do FreeRTOS

Para este tutorial, o arquivo de configuração do FreeRTOS está localizado em *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h. (Por exemplo, se AFR_BOARD espressif.esp32_devkitc for escolhido, o arquivo de configuração estará localizado em *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

> **Important**
>
> O dispositivo ATECC608A tem uma inicialização única que está bloqueada no dispositivo na primeira vez que um projeto é executado (durante a chamada para C_InitToken). No entanto, o projeto de demonstração e o projeto de teste do FreeRTOS têm configurações diferentes. Se o dispositivo estiver bloqueado durante as configurações do projeto de demonstração, nem todos os testes no projeto de teste serão bem-sucedidos.

1. Configure o projeto de demonstração do FreeRTOS seguindo as etapas em Configuração das demonstrações do FreeRTOS. Quando você chegar à última etapa Para formatar suas AWS IoT credenciais, pare e execute as etapas a seguir.
2. A Microchip forneceu várias ferramentas de script para ajudar a configurar as peças do ATECC608A. Navegue até o diretório *freertos*`/vendors/microchip/example_trust_chain_tool` e abra o arquivo `README.md`.
3. Siga as instruções no arquivo `README.md` para provisionar seu dispositivo. As etapas incluem o seguinte:
    1. Crie e registre uma autoridade de certificação com AWS.
    2. Gerar suas chaves no ATECC608A e exportar a chave pública e o número de série do dispositivo.
    3. Gere um certificado para o dispositivo e registre esse certificado com AWS.
4. Carregar o certificado CA e o certificado de dispositivo no dispositivo seguindo as instruções para Provisionamento de chaves no modo de desenvolvedor.

Monitorando mensagens MQTT na nuvem AWS

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console para monitorar AWS IoT as mensagens que seu dispositivo envia para a nuvem. AWS

Para assinar o tópico MQTT com o cliente AWS IoT MQTT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha Cliente de teste MQTT.
3. Em Tópico de assinatura, insira *your-thing-name*`/example/topic` e selecione Assinar o tópico.

Compilação, atualização e execução do projeto de demonstração do FreeRTOS usando o script idf.py

Você pode usar o utilitário Espressif (`idf.py`) do IDF para gerar os arquivos de compilação, compilar o binário da aplicação e instalar os binários em seu dispositivo.

> **Note**
>
> Algumas configurações podem exigir que você use a opção de porta "`-p port-name`" com`idf.py` para especificar a porta correta, como no exemplo a seguir.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Compilação e instalação do FreeRTOS no Windows, Linux e macOS (ESP-IDF v4.2)

1. Navegue até a raiz do diretório de downloads do FreeRTOS.
2. Na janela da linha de comando, insira o comando a seguir para adicionar as ferramentas de ESP-IDF ao PATH do seu terminal:

   Windows (aplicação "Comando")

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (aplicação "ESP-IDF 4.x CMD")

   (Isso já foi feito quando você abriu a aplicação.)

   Linux / macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Configure o cmake no diretório `build` e compile a imagem do firmware com o comando a seguir.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    build
   ```

   A saída deverá ser parecida com a do exemplo a seguir.

   ```
     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'
```

Se não houver erros, a compilação gerará os arquivos .bin binários do firmware.

4. Apague a memória instalada da placa de desenvolvimento com o comando a seguir.

```
idf.py erase_flash
```

5. Use o script `idf.py` para instalar o binário da aplicação na placa.

```
idf.py flash
```

6. Monitore a saída da porta serial da placa com o comando a seguir.

```
idf.py monitor
```

> **Note**
>
> - Você pode combinar esses comandos, como no exemplo a seguir.
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - Para determinadas configurações da máquina host, você deve especificar a porta ao instalar a placa, como no exemplo a seguir.
>
>   ```
>   idf.py erase_flash flash monitor -p /dev/ttyUSB1
>   ```

Compilação e instalação do FreeRTOS com o CMake

Além de usar o script `idf.py` fornecido pelo SDK do IDF para criar e executar seu código, você também pode compilar o projeto com o CMake. Atualmente, ele é compatível com o Makefile da Unix e o sistema de compilação Ninja.

Como compilar e instalar o projeto

1. Em uma janela da linha de comando, navegue até o diretório raiz de downloads do FreeRTOS.
2. Execute o script a seguir para adicionar as ferramentas ESP-IDF ao PATH do shell.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux / macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Insira o comando a seguir para gerar os arquivos de compilação.

   Com Makefiles da Unix

   ```
   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
   ```

   Com 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. Apague a instalada e depois instale a placa.

   Com Makefiles da Unix

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

Com Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

Mais informações

Para obter mais informações sobre o uso e a solução de problemas das placas Espressif ESP32, consulte os seguintes tópicos:

- Uso do FreeRTOS em seu próprio projeto CMake para ESP32
- Solução de problemas
- Depuração

Conceitos básicos do Espressif ESP32-S2

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

> **Note**
>
> Para explorar como integrar bibliotecas modulares e demonstrações do FreeRTOS no seu projeto Espressif IDF, consulte nossa Integração de referência em destaque para a plataforma ESP32-C3.

Este tutorial mostra como começar a usar as placas de desenvolvimento Espressif ESP32-S2 SoC e ESP32-S2-Saola-1.

Visão geral

Este tutorial orienta você pelas seguintes etapas:

1. Conectar a placa a uma máquina host.
2. Instalar o software na máquina host para desenvolver e depurar aplicações incorporadas para a placa do microcontrolador.
3. Fazer a compilação cruzada de uma aplicação de demonstração do FreeRTOS para uma imagem binária.
4. Carregar a imagem binária do aplicativo em na placa e executar o aplicativo.
5. Monitorar e depurar a aplicação em execução usando uma conexão serial.

Pré-requisitos

Antes de começar a usar os FreeRTOS em seu quadro Espressif, você deve configurar sua conta e permissões. AWS

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

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

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

Para conceder acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos em AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Criação de um conjunto de permissões no Guia do usuário do AWS IAM Identity Center .

- Usuários gerenciados no IAM com provedor de identidades:

  Crie um perfil para a federação de identidades. Siga as instruções em Criar um perfil para um provedor de identidades de terceiros (federação) no Guia do usuário do IAM.

- Usuários do IAM:
  - Crie um perfil que seu usuário possa assumir. Siga as instruções em Criação de um perfil para um usuário do IAM no Guia do usuário do IAM.
  - (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Conceitos básicos

> **Note**
>
> Os comandos do Linux neste tutorial exigem o uso do shell Bash.

1. Configuração de hardware da Espressif.

Para obter informações sobre como configurar o hardware da placa de desenvolvimento ESP32-S2, consulte o Guia de conceitos básicos do ESP32-S2-Saola-1.

> **Important**
>
> Ao chegar na seção Começar a usar dos guias do Espressif, pare e retorne para as instruções desta página.

2. Faça o download dos Amazon GitHubFreeRTOS em. (Para obter instruções, consulte o arquivo README.md.)
3. Configure o ambiente de desenvolvimento.

   Para se comunicar com sua placa, você deve instalar uma cadeia de ferramentas. A Espressif fornece o ESP-IDF para desenvolver software para suas placas. Como o ESP-IDF tem a própria versão do kernel do FreeRTOS integrada como um componente, o Amazon FreeRTOS inclui uma versão personalizada do ESP-IDF v4.2 que tem o kernel do FreeRTOS removido. Isso corrige problemas com arquivos duplicados durante a compilação. Para usar a versão personalizada do ESP-IDF v4.2 incluída no Amazon FreeRTOS, siga as instruções abaixo para o sistema operacional da sua máquina host.

   Windows

   1. Faça download do Instalador online universal do ESP-IDF para Windows.
   2. Execute o Instalador online universal.
   3. Ao chegar à etapa Fazer download ou usar ESP-IDF, selecione Usar um diretório ESP-IDF existente e defina Escolher diretório ESP-IDF existente como ***freertos*****/vendors/espressif/esp-idf**.
   4. Concluir a instalação.

   macOS

   1. Siga as instruções nos Pré-requisitos de configuração padrão da cadeia de ferramentas (ESP-IDF v4.2) para macOS.

> **Important**
>
> Quando você chegar nas instruções de "Obter ESP-IDF" em Próximas etapas, pare e retorne para as instruções desta página.

2. Abra a janela de linha de comando.
3. Navegue até o diretório de download do FreeRTOS e, em seguida, execute o script a seguir para baixar e instalar a cadeia de ferramentas do espressif em sua plataforma.

```
vendors/espressif/esp-idf/install.sh
```

4. Adicione as ferramentas da cadeia de ferramentas ESP-IDF ao caminho do seu terminal com o comando a seguir.

```
source vendors/espressif/esp-idf/export.sh
```

Linux

1. Siga as instruções nos Pré-requisitos da configuração padrão da cadeia de ferramentas (ESP-IDF v4.2) para Linux.

> **Important**
>
> Quando você chegar nas instruções de "Obter ESP-IDF" em Próximas etapas, pare e retorne para as instruções desta página.

2. Abra a janela de linha de comando.
3. Navegue até o diretório de download do FreeRTOS e execute o script a seguir para baixar e instalar a cadeia de ferramentas do Espressif em sua plataforma.

```
vendors/espressif/esp-idf/install.sh
```

4. Adicione as ferramentas da cadeia de ferramentas ESP-IDF ao caminho do seu terminal com o comando a seguir.

```
source vendors/espressif/esp-idf/export.sh
```

4. Estabelecimento de uma conexão serial.

   a. Para estabelecer uma conexão serial entre sua máquina host e o DevKit ESP32-C, instale os drivers CP210x USB para UART Bridge VCP. Você pode fazer download desses drivers no Silicon Labs.

   b. Siga as etapas para Estabelecer uma conexão serial com ESP32.

   c. Depois de estabelecer uma conexão serial, anote a porta serial da conexão de sua placa. Você precisa disso para instalar a demonstração.

Configuração das aplicações de demonstração do FreeRTOS

Para este tutorial, o arquivo de configuração do FreeRTOS está localizado em *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h. (Por exemplo, se AFR_BOARD espressif.esp32_devkitc for escolhido, o arquivo de configuração estará localizado em *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

1. Se você estiver executando macOS ou Linux, abra um prompt de terminal. Se você estiver executando o Windows, abra a aplicação "ESP-IDF 4.x CMD" (se você incluiu essa opção ao instalar a cadeia de ferramentas ESP-IDF) ou, caso contrário, a aplicação "Prompt de comando".
2. Para verificar se você tem o Python3 instalado, execute o seguinte:

   ```
   python --version
   ```

   A versão instalada é exibida. Se você não tiver o Python 3.0.1 ou posterior instalado, poderá instalá-lo a partir do site do Python.
3. Você precisa da interface de linha de AWS comando (CLI) para executar AWS IoT comandos. Se você estiver executando o Windows, use o easy_install awscli comando para instalar a AWS CLI no aplicativo “Command” ou “ESP-IDF 4.x CMD”.

   Se você estiver executando o macOS ou o Linux, consulte Instalação da CLI AWS.
4. Executar

   ```
   aws configure
   ```

   e configure a AWS CLI com seu ID de chave de AWS acesso, chave de acesso secreta e região padrão AWS . Para obter mais informações, consulte Configurar a CLI AWS.

5. Use o comando a seguir para instalar o AWS SDK para Python (boto3):

   - No Windows, na aplicação "Comando" ou "ESP-IDF 4.x CMD", execute

     ```
     easy_install boto3
     ```

   - No macOS ou Linux, execute

     ```
     pip install tornado nose --user
     ```

     e depois execute

     ```
     pip install boto3 --user
     ```

O FreeRTOS inclui o script `SetupAWS.py` para facilitar a configuração da placa Espressif para conectar-se ao AWS IoT.

Para executar o script de configuração

1. Para configurar o script, abra *`freertos`*`/tools/aws_config_quick_start/configure.json` e defina os seguintes atributos:

   `afr_source_dir`

   O caminho completo para o diretório *`freertos`* no computador. Certifique-se de usar barras para especificar esse caminho.

   `thing_name`

   O nome que você deseja atribuir à AWS IoT coisa que representa seu quadro.

   `wifi_ssid`

   O SSID da rede Wi-Fi.

   `wifi_password`

   A senha da rede Wi-Fi.

   `wifi_security`

   O tipo de segurança da rede Wi-Fi. Os tipos de segurança válidos estão a seguir:

- `eWiFiSecurityOpen` (Aberto, sem segurança)
- `eWiFiSecurityWEP` (segurança WEP)
- `eWiFiSecurityWPA` (segurança WPA)
- `eWiFiSecurityWPA2` (segurança WPA2)

2. Se você estiver executando macOS ou Linux, abra um prompt de terminal. Se você estiver executando o Windows, abra a aplicação "ESP-IDF 4.x CMD" ou "Comando".
3. Navegue até o diretório *`freertos`*`/tools/aws_config_quick_start` e execute

```
python SetupAWS.py setup
```

O script faz o seguinte:

- Cria AWS IoT algo, certificado e política.
- Anexa a AWS IoT política ao certificado e o certificado à AWS IoT coisa.
- Preenche o arquivo `aws_clientcredential.h` com o endpoint, o SSID Wi-Fi e as credenciais da AWS IoT .
- Formata o certificado e a chave privada e os grava no arquivo de cabeçalho `aws_clientcredential_keys.h`.

> **Note**
>
> O certificado é codificado apenas para fins de demonstração. Por este motivo, as aplicações devem armazenar esses arquivos em um local seguro.

Para obter mais informações sobre `SetupAWS.py`, consulte `README.md` no diretório *`freertos`*`/tools/aws_config_quick_start`.

Monitorando mensagens MQTT na nuvem AWS

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console para monitorar AWS IoT as mensagens que seu dispositivo envia para a nuvem. AWS

Para assinar o tópico MQTT com o cliente AWS IoT MQTT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha Cliente de teste MQTT.
3. Em Tópico de inscrição, insira *your-thing-name*`/example/topic` e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Compilação, atualização e execução do projeto de demonstração do FreeRTOS usando o script idf.py

Você pode usar o utilitário Espressif do IDF para gerar os arquivos de compilação, compilar o binário da aplicação e instalar a placa.

Compilação e instalação do FreeRTOS no Windows, Linux e macOS (ESP-IDF v4.2)

Use o script `idf.py` para compilar o projeto e instalar os binários em seu dispositivo.

> **Note**
>
> Algumas configurações podem exigir que você use a opção de porta `-p port-name` com`idf.py` para especificar a porta correta, como no exemplo a seguir.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Como compilar e instalar o projeto

1. Navegue até a raiz do diretório de downloads do FreeRTOS.
2. Na janela da linha de comando, insira o comando a seguir para adicionar as ferramentas de ESP-IDF ao PATH do seu terminal:

   Windows (aplicação "Comando")

   ```
   vendors\espressif\esp-idf\export.bat
   ```

Windows (aplicação "ESP-IDF 4.x CMD")

(Isso já foi feito quando você abriu a aplicação.)

Linux / macOS

```
source vendors/espressif/esp-idf/export.sh
```

3. Configure o cmake no diretório `build` e compile a imagem do firmware com o comando a seguir.

```
idf.py -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
```

A saída deverá ser parecida com a do exemplo a seguir.

```
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'
```

Se não houver erros, a compilação gera os arquivos.bin binários do firmware.

4. Apague a memória instalada da placa de desenvolvimento com o comando a seguir.

```
idf.py erase_flash
```

5. Use o script `idf.py` para instalar o binário da aplicação na placa.

```
idf.py flash
```

6. Monitore a saída da porta serial da placa com o comando a seguir.

```
idf.py monitor
```

> **Note**
>
> - Você pode combinar esses comandos, como no exemplo a seguir.
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - Para determinadas configurações da máquina host, você deve especificar a porta ao instalar a placa, como no exemplo a seguir.
>
>   ```
>   idf.py erase_flash flash monitor -p /dev/ttyUSB1
>   ```

Compilação e instalação do FreeRTOS com o CMake

Além de usar o script `idf.py` fornecido pelo SDK do IDF para criar e executar seu código, você também pode compilar o projeto com o CMake. Atualmente, ele é compatível com o Makefile da Unix e o sistema de compilação Ninja.

Como compilar e instalar o projeto

1. Em uma janela da linha de comando, navegue até o diretório raiz de downloads do FreeRTOS.
2. Execute o script a seguir para adicionar as ferramentas ESP-IDF ao PATH do shell.

   - Windows

     ```
     vendors\espressif\esp-idf\export.bat
     ```

   - Linux / macOS

```
source vendors/espressif/esp-idf/export.sh
```

3. Insira o comando a seguir para gerar os arquivos de compilação.

   - Com Makefiles da Unix

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
   B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
   ```

   - Com 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. Crie o projeto.

   - Com Makefiles da Unix

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

   - Com Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY -j8
   ```

5. Apague a instalada e depois instale a placa.

   - Com Makefiles da Unix

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   - Com Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Mais informações

Para obter mais informações sobre o uso e a solução de problemas das placas Espressif ESP32, consulte os seguintes tópicos:

- Uso do FreeRTOS em seu próprio projeto CMake para ESP32
- Solução de problemas
- Depuração

Introdução ao kit de conectividade IoT Infineon XMC48 00

> **Important**
>
> Essa integração de referência está hospedada no RTOS repositório Amazon-Free, que está obsoleto. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um RTOS projeto gratuito existente com base no repositório Amazon-FreeRTOS, agora obsoleto, consulte o. Guia de migração do repositório Github do Amazon FreeRTOS

Este tutorial fornece instruções para começar a usar o Infineon XMC48 00 IoT Connectivity Kit. Se você não tiver o kit de conectividade IoT Infineon XMC48 00, visite AWS o Catálogo de dispositivos do parceiro para comprar um de nosso parceiro.

Se você quiser abrir uma conexão serial com a placa para visualizar as informações de registro e depuração, precisará de um conversor de 3,3 USB V/serial, além do kit de conectividade 00 IoT. XMC48 O CP21 04 é um conversor USB /serial comum que está amplamente disponível em placas como a 04 Friend da Adafruit. CP21

Antes de começar, você deve configurar AWS IoT e RTOS fazer o download gratuito para conectar seu dispositivo à AWS nuvem. Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de RTOS download gratuito é chamado de*`freertos`*.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.

2. Compilação cruzada de um aplicativo de RTOS demonstração gratuito em uma imagem binária.
3. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.
4. Interagir com o aplicativo em execução na placa em uma conexão serial para fins de monitoramento e depuração.

Configuração do ambiente de desenvolvimento

Free RTOS usa o ambiente de DAVE desenvolvimento da Infineon para programar o XMC48 00. Antes de começar, você precisa baixar DAVE e instalar alguns drivers J-Link para se comunicar com o depurador integrado.

Instalar DAVE

1. Acesse a página de download do DAVE software da Infineon.
2. Escolha o DAVE pacote para seu sistema operacional e envie suas informações de registro. Depois de registrar na Infineon, você receberá um e-mail de confirmação com um link para fazer download de um arquivo .zip.
3. Baixe o arquivo.zip do DAVE pacote (`DAVE_version_os_date.zip`) e descompacte-o no local em que você deseja instalar DAVE (por exemplo,`C:\DAVE4`).

   > **Note**
   >
   > Alguns usuários do Windows relataram problemas usando o Windows Explorer para descompactar o arquivo. Recomendamos que você use um programa de terceiros como o 7-Zip.

4. Para iniciarDAVE, execute o arquivo executável encontrado na pasta descompactada`DAVE_version_os_date.zip`.

Para obter mais informações, consulte o Guia de início DAVE rápido.

Instalação do drivers Segger J-Link

Para se comunicar com a sonda de depuração CAT integrada da placa XMC48 00 Relax Ether, você precisa dos drivers incluídos no pacote de software e documentação do J-Link. Você pode fazer download do pacote e documentação J-Link na página de download do software J-Link da Segger.

Estabelecimento de uma conexão serial

A configuração de uma conexão serial é opcional, mas recomendada. Uma conexão serial permite que sua placa envie informações de registro e depuração em um formato que pode ser visualizado na sua máquina de desenvolvimento.

O aplicativo de demonstração XMC48 00 usa uma conexão UART serial nos pinos P0.0 e P0.1, que são rotulados na serigrafia da placa XMC48 00 Relax CAT Ether. Para configurar uma conexão serial:

1. Conecte o pino “RX <P0.0” ao pino “TX” do conversor USB /Serial.
2. Conecte o pino “TX>P0.1” ao pino “RX” do conversor USB /Serial.
3. Conecte o pino de aterramento do conversor serial a um dos pinos rotulados como “GND” na sua placa. Os dispositivos devem compartilhar um aterramento comum.

A alimentação é fornecida pela porta de USB depuração, portanto, não conecte o pino de voltagem positiva do adaptador serial à placa.

> **Note**
>
> Alguns cabos seriais usam um nível de sinalização 5V. A placa XMC48 00 e o módulo Wi-Fi Click requerem 3,3V. Não use o IOREF jumper da placa para alterar os sinais da placa para 5V.

Com o cabo conectado, você pode abrir uma conexão serial em um emulador de terminal, como o GNUScreen. A taxa de baud é definida como 115200 por padrão com 8 bits de dados, sem paridade e 1 bit de parada.

Monitorando MQTT mensagens na nuvem

Antes de executar a RTOS demonstração gratuita, você pode configurar o MQTT cliente no AWS IoT console para monitorar as mensagens que seu dispositivo envia para a AWS nuvem.

Para assinar o MQTT tópico com o AWS IoT MQTT cliente

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Testar e, em seguida, escolha cliente MQTT de teste para abrir o MQTT cliente.

3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Crie e execute o projeto de RTOS demonstração gratuito

Importe a RTOS demonstração gratuita para DAVE

1. ComeçarDAVE.
2. No DAVE, escolha File (Arquivo), Import (Importar). Na janela Importar, expanda a pasta Infineon, escolha DAVEProjeto e, em seguida, escolha Avançar.
3. Na janela Importar DAVE projetos, escolha Selecionar diretório raiz, escolha Procurar e, em seguida, escolha o projeto de demonstração XMC48 00.

   No diretório em que você descompactou o RTOS download gratuito, o projeto de demonstração está localizado em. `projects/infineon/xmc4800_iotkit/dave4/aws_demos`

   Verifique se a opção Copy Projects Into Workspace (Copiar projetos para o WorkSpace) está desmarcada.
4. Escolha Terminar.

   O projeto `aws_demos` deve ser importado para seu espaço de trabalho e ativado.
5. No menu Project (Projeto), escolha Build Active Project (Criar projeto ativo).

   Certifique-se de que o projeto seja criado sem erros.

Execute o projeto de RTOS demonstração gratuito

1. Use um USB cabo para conectar seu kit de conectividade XMC48 00 IoT ao seu computador. A placa tem dois USB microconectores. Use o "X101", onde Debug aparece próximo a ele no silkscreen da placa.
2. No menu Project (Projeto), escolha Rebuild Active Project (Recriar projeto ativo) para recriar `aws_demos` e garantir que as alterações de configuração sejam selecionadas.
3. No Project Explorer, clique com o botão direito do mouse`aws_demos`, escolha Depurar como e escolha Aplicativo DAVEC/C++.

4. Clique duas vezes em GDBSEGGERJ-Link Debugging para criar uma confirmação de depuração. Escolha Debug (Depurar).
5. Quando o depurador parar no ponto de interrupção em `main()`, no menu Run (Executar), escolha Resume (Continuar).

No AWS IoT console, o MQTT cliente das etapas 4-5 deve exibir as MQTT mensagens enviadas pelo seu dispositivo. Se você usar a conexão serial, verá algo parecido com isso na UART saída:

```
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----
```

Crie a RTOS demonstração gratuita com CMake

Se você preferir não usar um IDE RTOS desenvolvimento gratuito, você pode usar como alternativa CMake para criar e executar os aplicativos de demonstração ou aplicativos que você desenvolveu usando editores de código e ferramentas de depuração de terceiros.

> **Note**
>
> Esta seção aborda o uso CMake no Windows com o MinGW como sistema de compilação nativo. Para obter mais informações sobre o uso CMake com outros sistemas operacionais e opções, consulteUso da CMake com o FreeRTOS. (MinGW é um ambiente de desenvolvimento minimalista para aplicações nativas do Microsoft Windows.)

Para criar a RTOS demonstração gratuita com CMake

1. Configure o GNU Arm Embedded Toolchain.

   a. Faça o download de uma versão do conjunto de ferramentas do Windows na página de download do conjunto de ferramentas Arm Embedded.

      > **Note**
      >
      > Recomendamos que você faça download de uma versão diferente de "8-2018-q4-major", devido a um bug relatado com o utilitário "objcopy" nessa versão.

   b. Abra o instalador do conjunto de ferramentas obtido por download e siga as instruções do assistente de instalação para instalar o conjunto de ferramentas.

> **Important**
>
> Na página final do assistente de instalação, selecione Add path to environment variable (Adicionar caminho à variável de ambiente) para adicionar o caminho do conjunto de ferramentas à variável de ambiente do caminho do sistema.

2. Instale CMake e MinGW.

   Para obter instruções, consulte CMakePré-requisitos.

3. Crie uma pasta para conter os arquivos de compilação gerados (*build-folder*).

4. Altere os diretórios para seu diretório de RTOS download gratuito (*freertos*) e use o comando a seguir para gerar os arquivos de compilação:

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-
folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. Altere os diretórios para o diretório de construção (*build-folder*) e use o seguinte comando para criar o binário:

```
cmake --build . --parallel 8
```

   Esse comando cria o binário de saída `aws_demos.hex` para o diretório de compilação.

6. Flash e execute a imagem com JLINK.

   a. Do diretório de compilação (*build-folder*), use os seguintes comandos para criar um script flash:

```
echo loadfile aws_demos.hex > flash.jlink
```

```
echo r >> flash.jlink
```

```
echo g >> flash.jlink
```

```
echo q >> flash.jlink
```

   b. Faça o flash da imagem usando o JLNIK executável.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

Os logs de aplicação devem estar visíveis por meio da conexão de série estabelecida com a placa.

Solução de problemas

Se ainda não o fez, certifique-se de configurar AWS IoT e fazer o RTOS download gratuito para conectar seu dispositivo à AWS nuvem. Para obter instruções, consulte Primeiras etapas.

Para obter informações gerais sobre solução de problemas sobre como começar a usar o FreeRTOS, consulteSolução de problemas de conceitos básicos.

Conceitos básicos do Infineon OPTIGA Trust X e do XMC4800 IoT Connectivity Kit

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar o Infineon OPTIGA Trust X Secure Element e o XMC4800 IoT Connectivity Kit. Em comparação com o tutorial Introdução ao kit de conectividade IoT Infineon XMC48 00, este guia mostra como fornecer credenciais seguras usando um Infineon OPTIGA Trust X Secure Element.

Você precisa do seguinte hardware:

1. Host MCU – Kit de conectividade IoT XMC4800 da Infineon, acesse o catálogo do AWS Partner Device para comprar um com o nosso parceiro.
2. Pacote de extensões de segurança:
   - Secure Element - Infineon OPTIGA Trust X.

     Acesse o catálogo do AWS Partner Device para comprá-los de nosso parceiro.

- Placa de personalização — Placa de personalização Infineon OPTIGA.
- Placa adaptadora — Adaptador Infineon MyIoT.

Para seguir as etapas aqui, você deve abrir uma conexão serial com a placa para visualizar as informações de registro e depuração. (Uma das etapas exige que você copie uma chave pública da saída de depuração serial da placa e cole-a em um arquivo.) Para fazer isso, será necessário ter um conversor USB/serial de 3.3 V, além do kit de conectividade IoT XMC4800. Sabe-se que o conversor USB/serial JBtek EL-PN-47310126 funciona para essa demonstração. Também são necessários três fios de jumper macho para macho (para recebimento (RX), transmissão (TX) e terra (GND)) para conectar o cabo serial à placa adaptadora MyIoT da Infineon.

Antes de começar, você deve configurar o AWS IoT e seu download do FreeRTOS para conectar seu dispositivo à nuvem da AWS. Para obter instruções, consulte Opção 2: geração de chaves privadas integradas. Neste tutorial, o caminho para o diretório de download do FreeRTOS é chamado *freertos*.

Visão geral

Este tutorial contém as seguintes etapas:

1. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para a placa do microcontrolador.
2. Fazer a compilação cruzada de um aplicativo de demonstração do FreeRTOS para uma imagem binária.
3. Carregar a imagem binária do aplicativo em na placa e executar o aplicativo.
4. Para fins de monitoramento e depuração, interagir com o aplicativo em execução na placa em uma conexão serial.

Configuração do ambiente de desenvolvimento

O FreeRTOS usa o ambiente de desenvolvimento DAVE da Infineon para programar o XMC4800. Antes de começar, faça download e instale o DAVE e alguns drivers J-Link para se comunicar com o depurador na placa.

Instalar o DAVE

1. Vá para a página de download do software DAVE da Infineon.

2. Escolha o pacote DAVE para seu sistema operacional e envie suas informações de registro. Após o registro, você receberá um e-mail de confirmação com um link para fazer download de um arquivo .zip.
3. Faça download do arquivo .zip do pacote DAVE (`DAVE_`*`version_os_date`*`.zip`) e descompacte-o no local onde você deseja instalar o DAVE (por exemplo, `C:\DAVE4`).

> **Note**
>
> Alguns usuários do Windows relataram problemas usando o Windows Explorer para descompactar o arquivo. Recomendamos que você use um programa de terceiros como o 7-Zip.

4. Para iniciar o DAVE, execute o arquivo executável encontrado na pasta `DAVE_`*`version_os_date`*`.zip` descompactado.

Para obter mais informações, consulte o Guia de início rápido do DAVE.

Instalação do drivers Segger J-Link

Para se comunicar com o teste de depuração integrado do XMC4800 IoT Connectivity kit, você precisa dos drivers incluídos no pacote de software e documentação J-Link. Você pode fazer download do pacote e documentação J-Link na página de download do software J-Link da Segger.

Estabelecimento de uma conexão serial

Conecte o cabo do conversor USB/serial ao adaptador Infineon Shield2Go. Uma conexão serial permite que sua placa envie informações de registro em log e depuração em um formato que pode ser visualizado na máquina de desenvolvimento. Para configurar uma conexão serial:

1. Conecte o pino RX ao pino TX do conversor USB/serial.
2. Conecte o pino TX ao pino RX do conversor USB/serial.
3. Conecte o pino terra do conversor serial a um dos pinos GND na sua placa. Os dispositivos devem compartilhar um aterramento comum.

A alimentação é fornecida na porta de depuração USB e, portanto, não conecto o pino de tensão positiva do adaptador serial à placa.

> **Note**
>
> Alguns cabos seriais usam um nível de sinalização 5V. A placa XMC4800 e o módulo Wi-Fi Click exigem 3,3 V. Não use o jumper IOREF da placa para alterar os sinais da placa para 5V.

Com o cabo conectado, você pode abrir uma conexão serial em um emulador de terminal como GNU Screen. A taxa de baud é definida como 115200 por padrão com 8 bits de dados, sem paridade e 1 bit de parada.

Monitoramento de mensagens MQTT na nuvem

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console do AWS IoT para monitorar as mensagens enviadas pelo dispositivo para a nuvem da AWS.

Para assinar o tópico MQTT com o cliente MQTT do AWS IoT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Compilação e execução do projeto de demonstração do FreeRTOS

Importação da demonstração do FreeRTOS para o DAVE

1. Inicie o DAVE.
2. No DAVE, escolha File (Arquivo) e depois Import (Importar). Expanda a pasta Infineon escolha DAVE Project (Projeto DAVE) e depois Next (Avançar).
3. Na janela Import DAVE Projects (Importar projetos do DAVE), escolha Select Root Directory (Selecionar diretório raiz), escolha Browse (Procurar) e, em seguida, escolha o projeto de demonstração do XMC4800.

No diretório em que você descompactou o download do FreeRTOS, o projeto de demonstração está localizado em `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`.

Verifique se a opção Copy Projects Into Workspace (Copiar projetos para o WorkSpace) está desmarcada.

4. Escolha Finish.

   O projeto `aws_demos` deve ser importado para seu espaço de trabalho e ativado.

5. No menu Project (Projeto), escolha Build Active Project (Criar projeto ativo).

   Certifique-se de que o projeto seja criado sem erros.

Execução do projeto de demonstração do FreeRTOS

1. No menu Project (Projeto), escolha Rebuild Active Project (Recriar projeto ativo) para reconstruir `aws_demos` e garantir que as alterações de configuração sejam selecionadas.
2. No Project Explorer, clique com o botão direito do mouse em `aws_demos`, escolha Debug As (Depurar como) e DAVE C/C++ Application (Aplicativo DAVE C/C++).
3. Clique duas vezes em GDB SEGGER J-Link Debugging (Depuração de GDB SEGGER J-Link) para criar uma confirmação de depuração. Escolha Debug (Depurar).
4. Quando o depurador parar no ponto de interrupção em `main()`, no menu Run (Executar), escolha Resume (Continuar).

Neste ponto, continue com a etapa de extração de chave pública em Opção 2: geração de chaves privadas integradas. Depois que todas as etapas estiverem concluídas, vá para o console do AWS IoT. O cliente MQTT configurado anteriormente deve exibir as mensagens MQTT enviadas pelo seu dispositivo. Por meio da conexão serial do dispositivo, você deve ver algo assim na saída 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----
```

Compilação da demonstração do FreeRTOS com CMake

Esta seção discute o uso do CMake no Windows com MingW como sistema de compilação nativa. Para obter mais informações sobre como usar CMake com outros sistemas operacionais e opções, consulte Uso da CMake com o FreeRTOS. (MinGW é um ambiente de desenvolvimento minimalista para aplicativos nativos do Microsoft Windows.)

Se você preferir não usar um IDE para desenvolvimento do FreeRTOS, você pode usar o CMake para compilar e executar os aplicativos de demonstração ou aplicativos que você tenha desenvolvido usando editores de código e ferramentas de depuração de terceiros.

Como compilar a demonstração do FreeRTOS com CMake

1. Configure o conjunto de ferramentas GNU Arm Embedded.

   a. Faça download de uma versão do conjunto de ferramentas do Windows na página de download do conjunto de ferramentas Arm Embedded.

   > **Note**
   >
   > Devido a um bug relatado no utilitário objcopy, recomendamos que você faça download de uma versão diferente de "8-2018-q4-major".

   b. Abra o instalador da cadeia de ferramentas transferido por download e siga as instruções no assistente.

   c. Na página final do assistente de instalação, selecione Add path to environment variable (Adicionar caminho à variável de ambiente) para adicionar o caminho do conjunto de ferramentas à variável de ambiente do caminho do sistema.

2. Instale o CMake e MingW.

   Para obter instruções, consulte Pré-requisitos do CMake.

3. Crie uma pasta para conter os arquivos de compilação gerados (*build-folder*).

4. Altere os diretórios para seu diretório de download do (*freertos*) e use o comando a seguir para gerenciar os arquivos de compilação:

   ```
   cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S .
    -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
   ```

5. Altere os diretórios para o diretório de compilação (*build-folder*) e use o seguinte comando para compilar o binário:

   ```
   cmake --build . --parallel 8
   ```

   Esse comando cria o binário de saída `aws_demos.hex` para o diretório de compilação.

6. Atualize e execute a imagem com JLINK.

   a. No diretório de compilação (*build-folder*), use os seguintes comandos para criar um script de atualização:

```
echo loadfile aws_demos.hex > flash.jlink
echo r >> flash.jlink
echo g >> flash.jlink
echo q >> flash.jlink
```

b. Atualize a imagem usando o executável JLNIK.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

Os logs de aplicativo devem estar visíveis por meio da conexão de série estabelecida com a placa. Continue para a etapa de extração de chave pública em Opção 2: geração de chaves privadas integradas. Depois que todas as etapas estiverem concluídas, vá para o console do AWS IoT. O cliente MQTT configurado anteriormente deve exibir as mensagens MQTT enviadas pelo seu dispositivo.

Solução de problemas

Para obter informações gerais sobre a solução de problemas, consulte Solução de problemas de conceitos básicos.

Introdução ao kit inicial MW32x AWS IoT

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

O AWS IoT Starter Kit é um kit de desenvolvimento baseado no 88MW320/88MW322, o mais recente microcontrolador Cortex M4 integrado da NXP, que integra Wi-Fi 802.11b/g/n em um único chip microcontrolador. O kit de desenvolvimento é certificado pela FCC. Para obter mais informações, consulte o catálogo do AWS Partner Device para comprar um de nosso parceiro. Os módulos 88MW320/88MW322 também são certificados pela FCC e estão disponíveis para personalização e venda por volume.

Este guia de conceitos básicos mostra como fazer a compilação cruzada da aplicação junto com o SDK em um computador host e, em seguida, carregar o arquivo binário gerado na placa usando as ferramentas fornecidas com o SDK. Quando a aplicação começa a ser executada na placa, é possível depurá-la ou interagir com ela a partir do console serial do computador host.

O Ubuntu 16.04 é a plataforma host compatível para desenvolvimento e depuração. Será possível usar outras plataformas, mas não há suporte oficial para elas. Você deve ter permissões para instalar o software na plataforma host. As ferramentas externas a seguir são necessárias para criar o SDK:

- Plataforma host do Ubuntu 16.04
- Cadeia de ferramentas do ARM versão 4_9_2015q3
- IDE do Eclipse 4.9.0

A cadeia de ferramentas do ARM é necessária para fazer a compilação cruzada da aplicação e do SDK. O SDK aproveita as versões mais recentes da cadeia de ferramentas para otimizar a marca da imagem e encaixar mais funcionalidades em menos espaço. Este guia considera que você esteja usando a versão 4_9_2015q3 da cadeia de ferramentas. Não é recomendável usar versões mais antigas da cadeia de ferramentas. O kit de desenvolvimento é pré-instalado com o firmware do projeto de demonstração do microcontrolador sem fios.

Tópicos

Configuração do hardware

Conecte a placa MW32x ao laptop usando um cabo mini-USB para USB. Conecte o cabo mini-USB ao único conector mini-USB presente na placa. Não é necessário alterar o jumper.

Se a placa estiver conectada a um laptop ou computador desktop, não será necessário obter uma fonte de alimentação externa.

Essa conexão USB fornece o seguinte:

- Console de acesso à placa. Uma porta tty/com virtual é registrada no host de desenvolvimento que pode ser usada para acessar o console.
- Acesso JTAG à placa. Isso pode ser usado para carregar ou descarregar imagens de firmware na memória RAM ou flash da placa, ou para fins de depuração.

Configuração do ambiente de desenvolvimento

Para fins de desenvolvimento, o requisito mínimo é a cadeia de ferramentas do ARM e as ferramentas fornecidas com o SDK. As seções a seguir fornecem detalhes sobre a configuração da cadeia de ferramentas do ARM.

Conjunto de ferramentas para GNU

O SDK oferece suporte oficial à cadeia de ferramentas do compilador GCC. A cadeia de ferramentas de compilação cruzada para GNU do ARM está disponível na cadeia de ferramentas integradas do ARM para GNU 4.9-2015-q3-update.

O sistema de compilação é configurado para usar a cadeia de ferramentas do GNU por padrão. Os Makefiles consideram os binários da cadeia de ferramentas do compilador GNU como disponíveis no CAMINHO do usuário e podem ser invocados a partir dos Makefiles. Os Makefiles também consideram os nomes dos arquivos dos binários da cadeia de ferramentas do GNU como prefixados com `arm-none-eabi-`.

A cadeia de ferramentas do GCC pode ser usada com o GDB para depurar com o OpenOCD (incluído no SDK). Isso fornece o software que interage com o JTAG.

Recomendamos a versão 4_9_2015q3 do conjunto de ferramentas. gcc-arm-embedded

Procedimento de configuração da cadeia de ferramentas do Linux

Siga estas etapas para configurar a cadeia de ferramentas do GCC no Linux.

1. Faça o download da cadeia de ferramentas tarball disponível na cadeia de ferramentas integradas do ARM para GNU 4.9-2015-q3-update. O arquivo é `gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`.
2. Copie o arquivo para um diretório de sua escolha. Não deve haver espaços no nome do diretório.
3. Use o comando a seguir para descompactar o arquivo.

```
tar -vxf filename
```

4. Adicione o caminho da cadeia de ferramentas instalada ao PATH do sistema. Por exemplo, anexe a linha a seguir no final do arquivo `.profile` localizado no diretório `/home/user-name`.

```
PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
```

> **Note**
>
> Novas distribuições do Ubuntu podem ter uma versão do Debian do Compilador cruzado do GCC. Nesse caso, você deve remover o Compilador cruzado nativo e seguir o procedimento de configuração acima.

Trabalho com um host de desenvolvimento do Linux

Toda distribuição moderna do desktop Linux, como o Ubuntu ou Fedora, pode ser usada. No entanto, recomendamos atualizar para a versão mais recente. As etapas a seguir foram verificadas para funcionar no Ubuntu 16.04 e consideram que você está usando essa versão.

Instalação de pacotes

O SDK inclui um script para permitir a configuração rápida do ambiente de desenvolvimento em uma máquina Linux configurada recentemente. O script tenta detectar o tipo de máquina e instalar o software adequado automaticamente, incluindo bibliotecas C, biblioteca USB, biblioteca FTDI, ncurses, python e latex. Nesta seção, o nome genérico do diretório *amzsdk_bundle-x.y.z* indica o diretório raiz do AWS SDK. O nome real do diretório pode ser diferente. Você deve ter privilégios de root.

- Navegue até o diretório *amzsdk_bundle-x.y.z/* e execute esse comando.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

Evitando o sudo

Neste guia, a operação `flashprog` usa o script `flashprog.py` para instalar o NAND da placa, conforme explicado abaixo. Da mesma forma, a operação `ramload` usa o script `ramload.py` para

copiar a imagem do firmware do host diretamente para a RAM do microcontrolador, sem instalar o NAND.

Você pode configurar seu host de desenvolvimento Linux para realizar as operações `flashprog` e `ramload` sem exigir o comando `sudo` todas as vezes. Para fazer isso, execute o comando a seguir.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

> **Note**
>
> Você deve configurar suas permissões de host de desenvolvimento Linux dessa forma para garantir uma experiência suave no IDE do Eclipse.

Configuração do console serial

Insira o cabo USB no slot USB do host Linux. Isso aciona a detecção do dispositivo. Você deve ver mensagens como as seguintes no arquivo `/var/log/messages` ou depois de executar o 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
```

Verifique se dois dispositivos ttyUSB foram criados. O segundo ttyUSB é o console de série. No exemplo acima, isso é chamado de "ttyUSB1".

Neste guia, usamos o minicom para ver a saída do console de série. Você também pode usar outros programas seriais, como `putty`. Execute o comando a seguir para executar o minicom no modo de configuração.

```
minicom -s
```

No minicom, navegue até Configuração de porta serial e capture as seguintes configurações.

```
| 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
```

Você pode salvar essas configurações no minicom para uso futuro. A janela do minicom agora exibe mensagens do console de série.

Escolha a janela do console de série e pressione a tecla Enter. Isso exibe um hash (#) na tela.

> **Note**
>
> As placas de desenvolvimento incluem um dispositivo de chip FTDI. O dispositivo FTDI expõe duas interfaces USB para o host. A primeira interface está associada à funcionalidade JTAG do MCU e a segunda interface está associada à porta física UARTx do MCU.

Instalação do OpenOCD

O OpenOCD é um software que fornece depuração, programação no sistema e teste de verificação de limites para dispositivos de destino incorporados.

É necessário ter o OpenOCD versão 0.9. Também é necessário para a funcionalidade do Eclipse. Se uma versão anterior, como a versão 0.7, foi instalada em seu host Linux, remova esse repositório com o comando apropriado para a distribuição Linux que você está usando atualmente.

Execute o comando padrão do Linux para instalar o OpenOCD,

```
apt-get install openocd
```

Se o comando acima não instalar a versão 0.9 ou posterior, use o procedimento a seguir para baixar e compilar o código-fonte do openocd.

Como instalar o OpenOCD

1. Execute o comando a seguir para instalar libusb-1.0.

```
sudo apt-get install libusb-1.0
```

2. Faça download do código-fonte do OpenOCD 0.9.0 em http://openocd.org/.
3. Extraia o OpenOCD e navegue até o diretório em que você o extraiu.
4. Configure o OpenOCD com o comando a seguir.

```
./configure --enable-ftdi --enable-jlink
```

5. Execute o utilitário make para compilar o OpenOCD.

```
make install
```

Configuração do Eclipse

> **Note**
>
> Esta seção considera que você concluiu as etapas em Evitando o sudo

O Eclipse é o IDE preferido para desenvolvimento e depuração de aplicações. Ele fornece um IDE avançado e fácil de usar com suporte à depuração integrado, incluindo depuração com reconhecimento de threads. Esta seção descreve a configuração comum do Eclipse para todos os hosts de desenvolvimento que são compatíveis.

Como configurar o Eclipse

1. Baixe e instale o Java Run Time Environment (JRE).

   O Eclipse exige a instalação do JRE. Recomendamos que você instale isso primeiro, embora ele possa ser instalado depois de instalar o Eclipse. A versão do JRE (32/64 bits) deverá corresponder à versão do Eclipse (32/64 bits). Você pode baixar o JRE em 8 Downloads do Java SE Runtime Environment no site da Oracle.
2. Baixe e instale o "IDE do Eclipse para desenvolvedores C/C++" do http://www.eclipse.org. A versão do Eclipse 4.9.0 ou posterior é compatível. A instalação exige apenas que você extraia

o arquivo baixado. Você executa o executável Eclipse específico da plataforma para iniciar a aplicação.

Compilação e execução do projeto de demonstração do FreeRTOS

Há duas maneiras de executar o projeto de demonstração do FreeRTOS:

- Usar a linha de comando.
- Abra o IDE do Eclipse.

Este tópico aborda as duas opções.

Provisionamento

- Dependendo se você usa a aplicação de teste ou de demonstração, defina os dados de provisionamento em um dos seguintes arquivos:

  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  Por exemplo: .

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

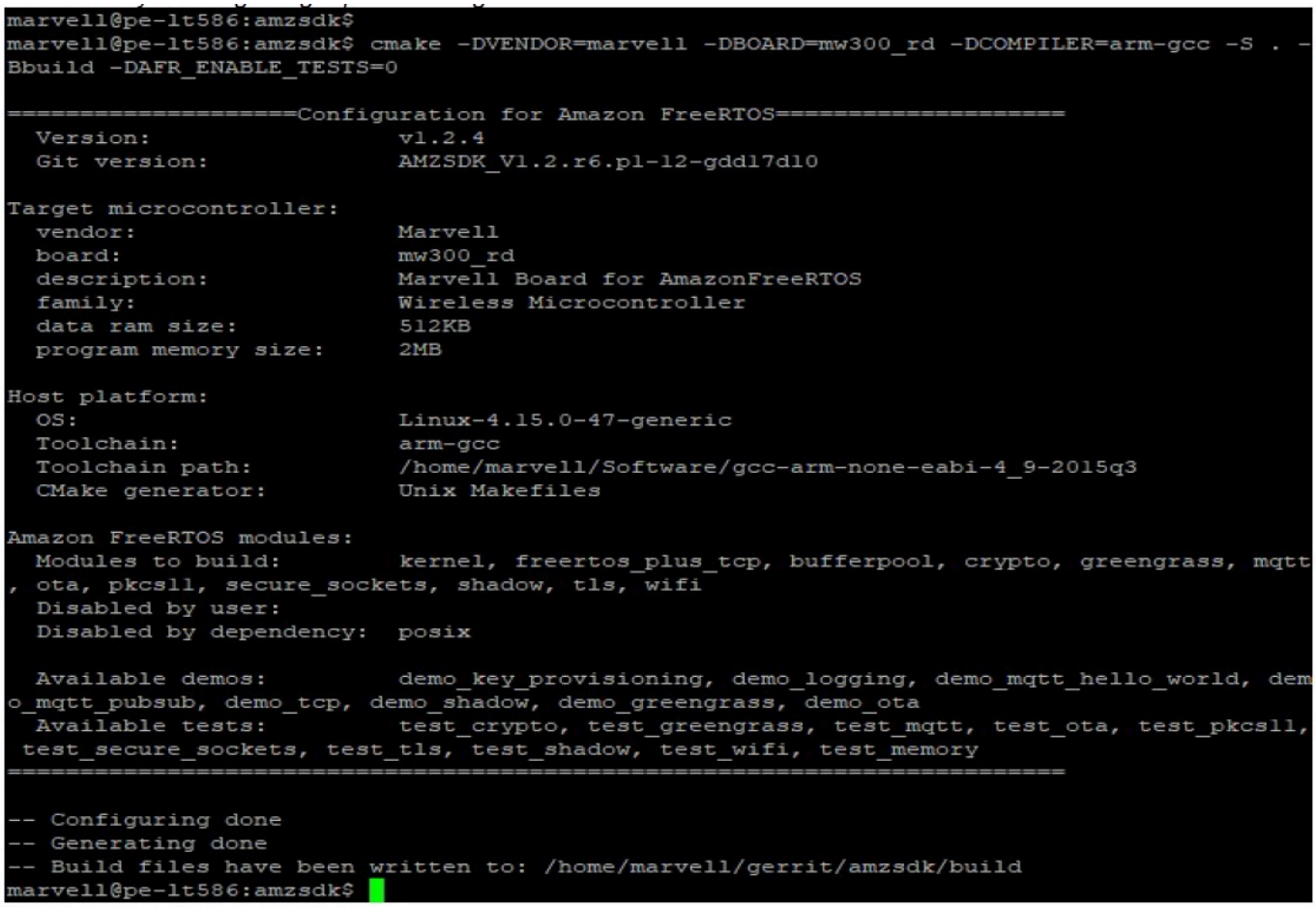
> **Note**
>
> Você pode inserir os seguintes valores de segurança de Wi-Fi:
>
> - `eWiFiSecurityOpen`
> - `eWiFiSecurityWEP`
> - `eWiFiSecurityWPA`
> - `eWiFiSecurityWPA2`
>
> O SSID e a senha devem ser colocados entre aspas duplas.

Compilação e execução da demonstração do FreeRTOS usando a linha de comando

1. Use o comando a seguir para começar a compilar a aplicação de demonstração.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=0
```

   Obtenha a mesma saída mostrada no exemplo a seguir.



2. Navegue até o diretório de compilação.

```
cd build
```

3. Execute o utilitário make para compilar a aplicação.

```
make all -j4
```

Obtenha a mesma saída mostrada na figura a seguir:

4. Use os comandos a seguir para compilar uma aplicação de teste.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=1
cd build
make all -j4
```

Execute o comando `cmake` toda vez que você alternar entre o `aws_demos project` e o `aws_tests project`.

5. Grave a imagem do firmware na instalação da placa de desenvolvimento. O firmware será executado após a redefinição da placa de desenvolvimento. Você deve criar o SDK antes de instalar a imagem para o microcontrolador.

   a. Antes de instalar a imagem do firmware, prepare a instalação da placa de desenvolvimento com os componentes comuns Layout e Boot2. Use os seguintes 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
   ```

   O comando `flashprog` inicia o seguinte:

- Layout: o utilitário flashprog é instruído primeiro a gravar um layout na instalação. O layout é semelhante às informações de partição da instalação. O layout padrão está localizado em `/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.
- Boot2: este é o carregador de inicialização usado pelo WMSDK. O script `flashprog` também grava um carregador de inicialização na instalação. O carregador de inicialização carrega a imagem do firmware do microcontrolador depois que ela é instalada na placa. Obtenha a mesma saída mostrada na figura abaixo.

```
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. O firmware usa o chipset Wi-Fi para sua funcionalidade, e o chipset Wi-Fi tem seu próprio firmware que também deve estar presente na instalação. Você usa o utilitário `flashprog.py` para atualizar o firmware Wi-Fi da mesma forma que fez para atualizar o carregador de inicialização Boot2 e o firmware do MCU. Use os comandos a seguir para instalar o firmware do 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
```

A saída do comando deve ser semelhante à figura abaixo.

```
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. Use os comandos a seguir para instalar o firmware da 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. Redefina a placa. Você deve ver logs para a aplicação de demonstração.

e. Para executar a aplicação de teste, atualize o binário `aws_tests.bin` localizado no mesmo diretório.

```
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
```

A saída de comando deve ser parecida com a mostrada na figura abaixo.

```
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. Depois de instalar o firmware e reiniciar a placa, a aplicação de demonstração deve iniciar conforme mostrado na figura abaixo.

```
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----
```

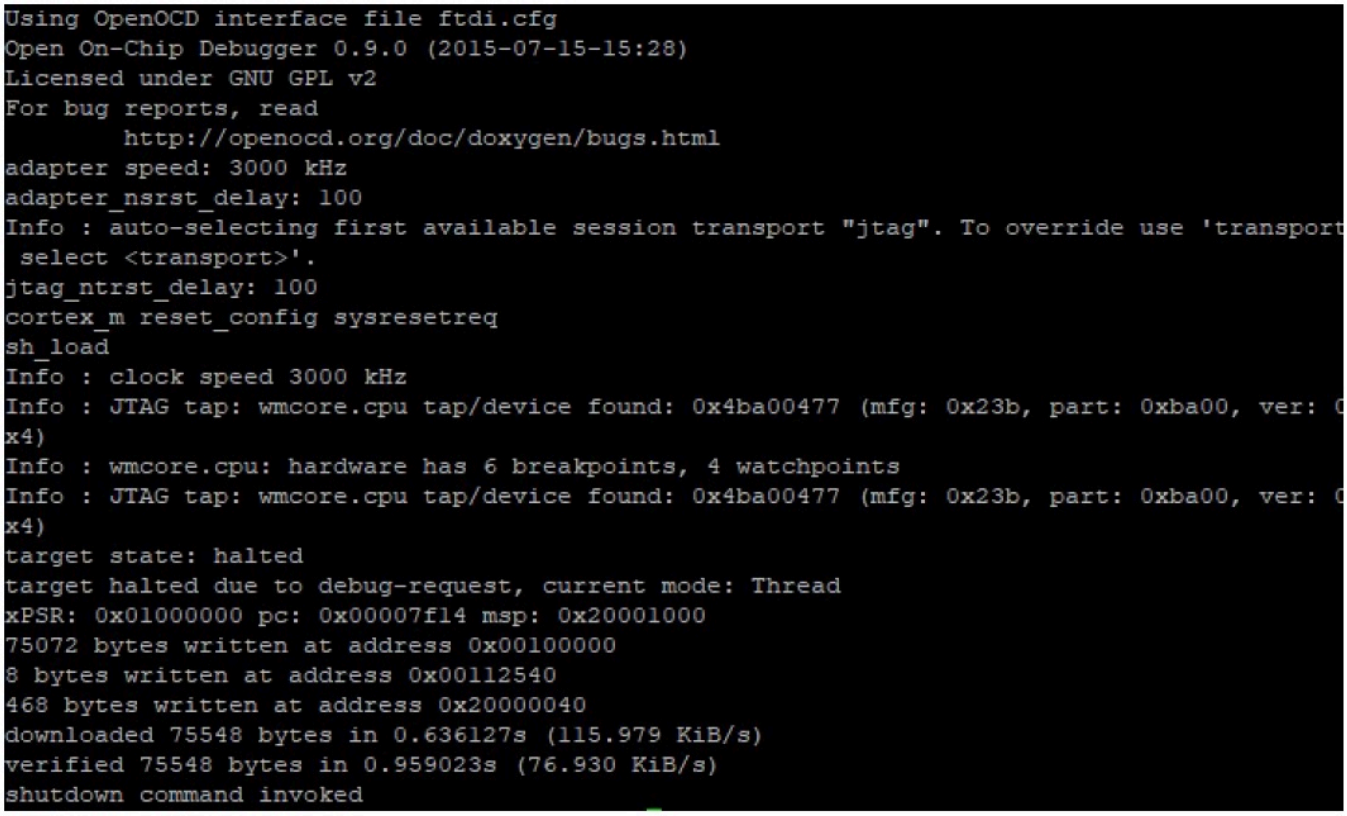
7. (Opcional) Como método alternativo para testar a imagem, use o utilitário flashprog para copiar a imagem do microcontrolador do host diretamente na RAM do microcontrolador. A imagem não é copiada na instalação, então ela será perdida após a reinicialização do microcontrolador.

   Carregar a imagem do firmware na SRAM é uma operação mais rápida porque inicia o arquivo de execução imediatamente. Esse método é usado principalmente para desenvolvimento iterativo.

   Use os comandos a seguir para carregar o firmware na 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
   ```

   A saída do comando é mostrada na figura abaixo.



Quando a execução do comando for concluída, você deverá ver os logs da aplicação de demonstração.

Compilação e execução da demonstração do FreeRTOS usando o IDE do Eclipse

1. Antes de configurar um espaço de trabalho do Eclipse, você deve executar o comando `cmake`.

   Execute o comando a seguir para trabalhar com o projeto Eclipse do `aws_demos`.

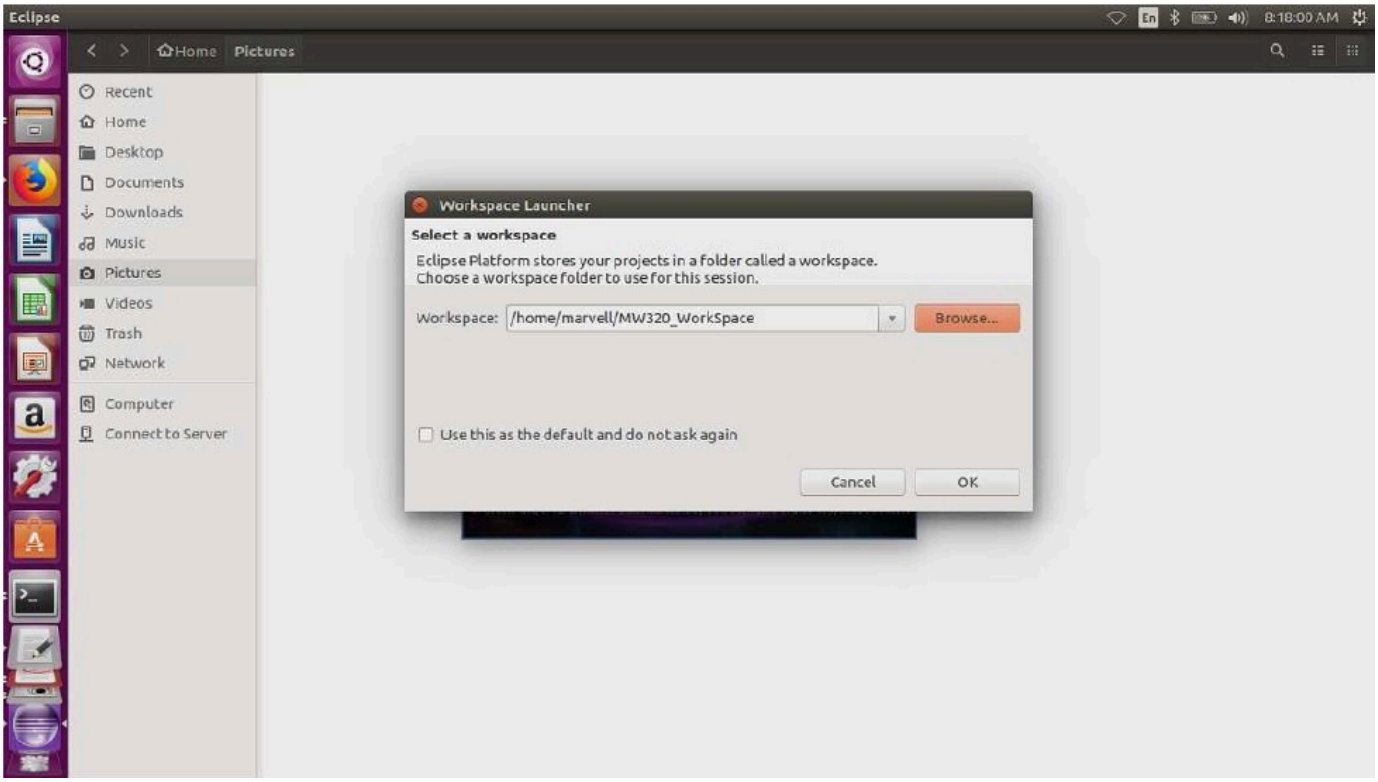
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Execute o comando a seguir para trabalhar com o projeto Eclipse do `aws_tests`.

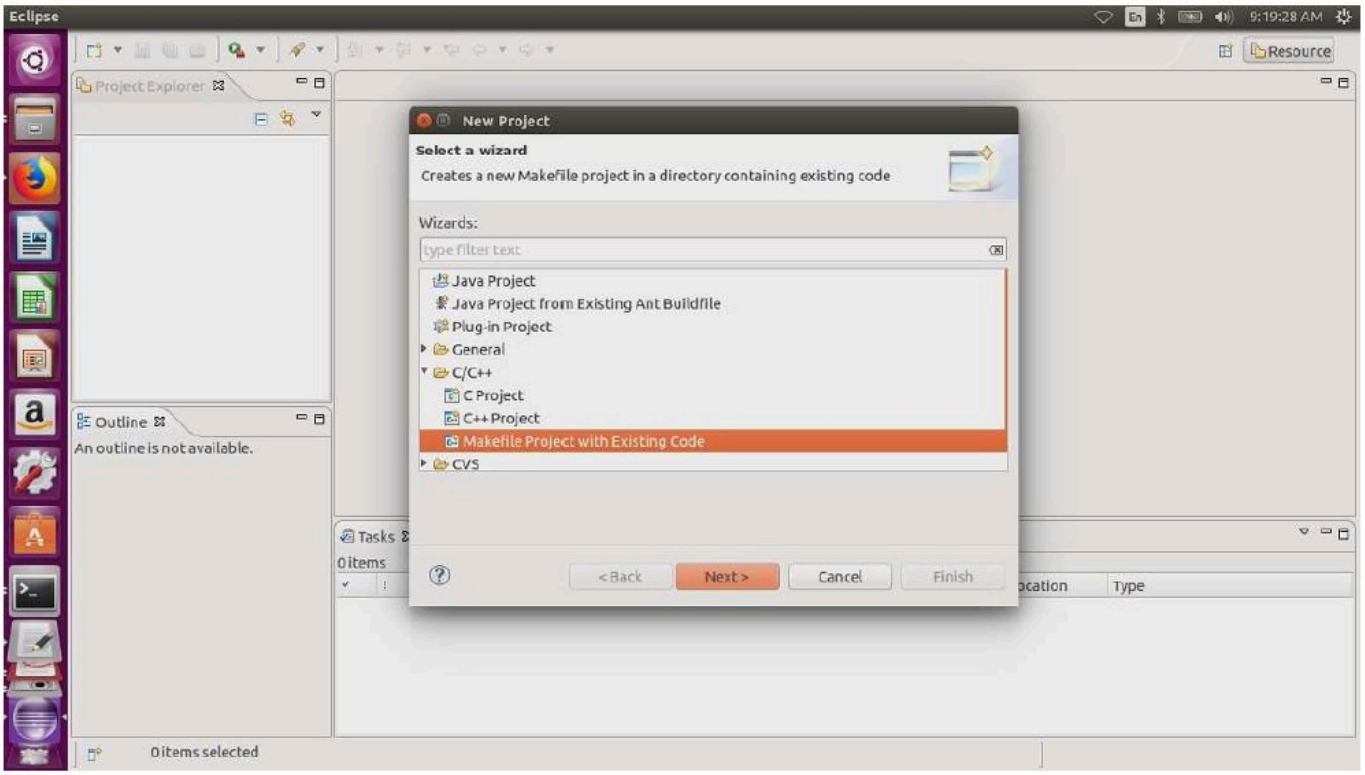
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> Execute o comando `cmake` toda vez que você alternar entre o projeto `aws_demos` e o `aws_tests`.

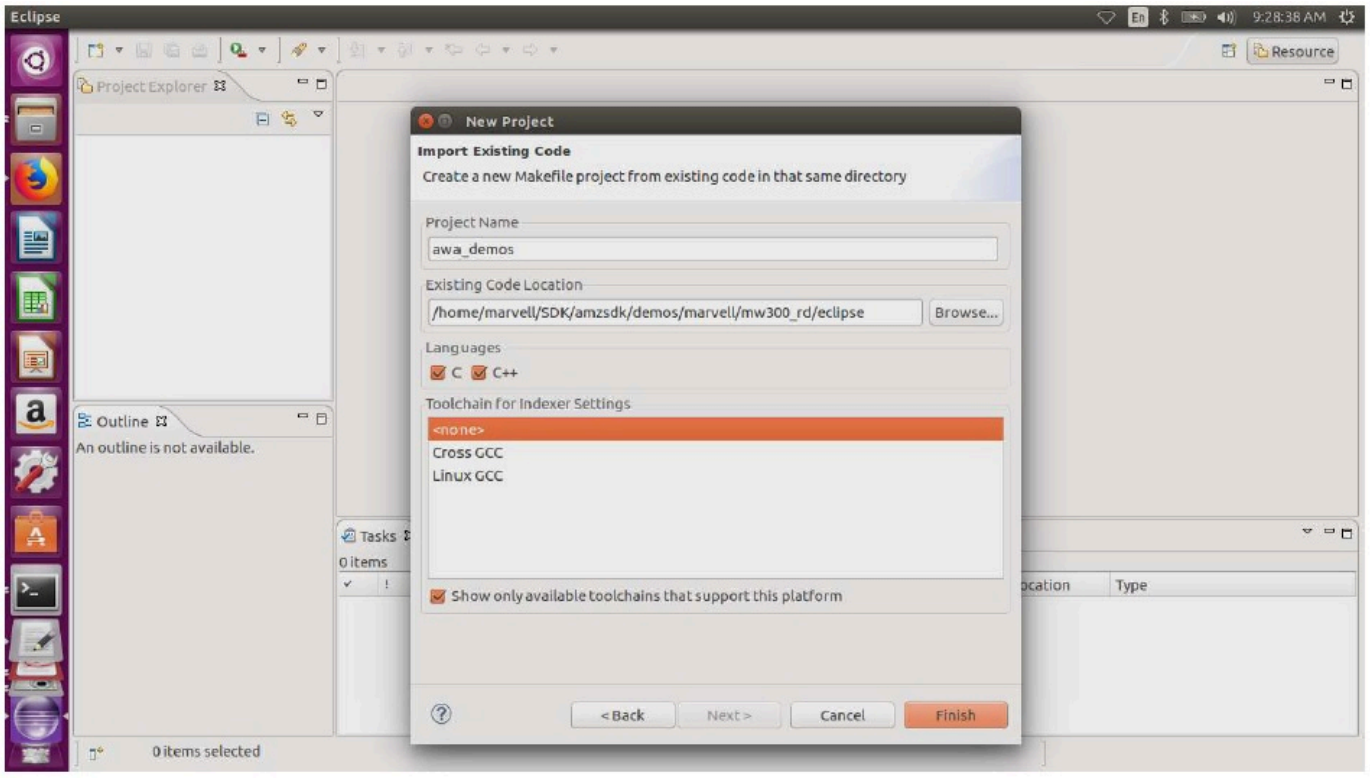
2. Abra o Eclipse e, quando solicitado, escolha a área de trabalho do Eclipse, conforme mostrado na figura abaixo.



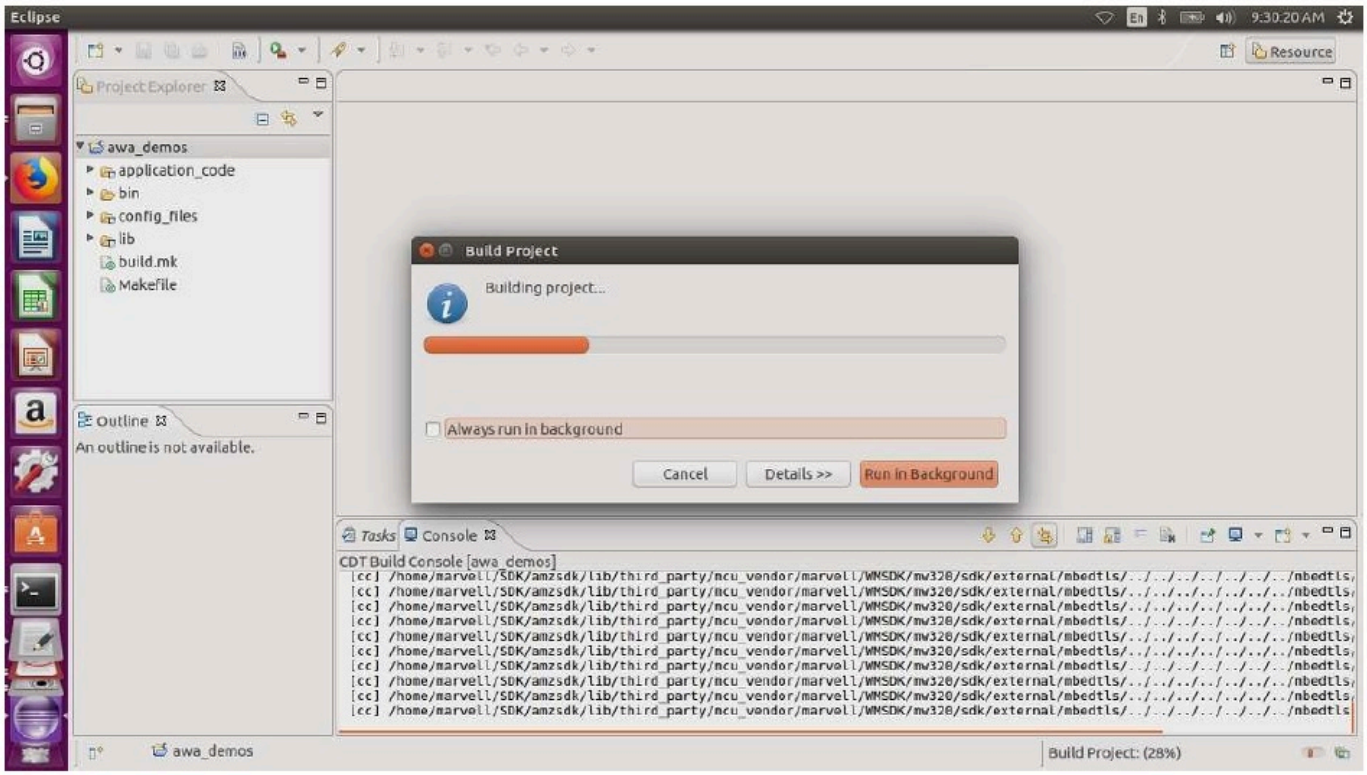
3. Escolha a opção de criar um projeto Makefile: com código existente, conforme mostrado na figura abaixo.



4. Escolha Procurar, especifique o diretório do código existente e escolha Concluir.



5. No painel de navegação, escolha aws_demos no explorador de projetos. Clique com o botão direito do mouse em aws_demos para abrir o menu e escolha Compilar.



Se houver êxito na compilação, ela gerará o arquivo `build/cmake/vendors/marvell/mw300_rd/aws_demos.bin`.

6. Use as ferramentas da linha de comando para atualizar o arquivo de layout (`layout.txt`), o binário Boot2 (`boot2.bin`), o binário do firmware MCU (`aws_demos.bin`) e o firmware Wi-Fi.

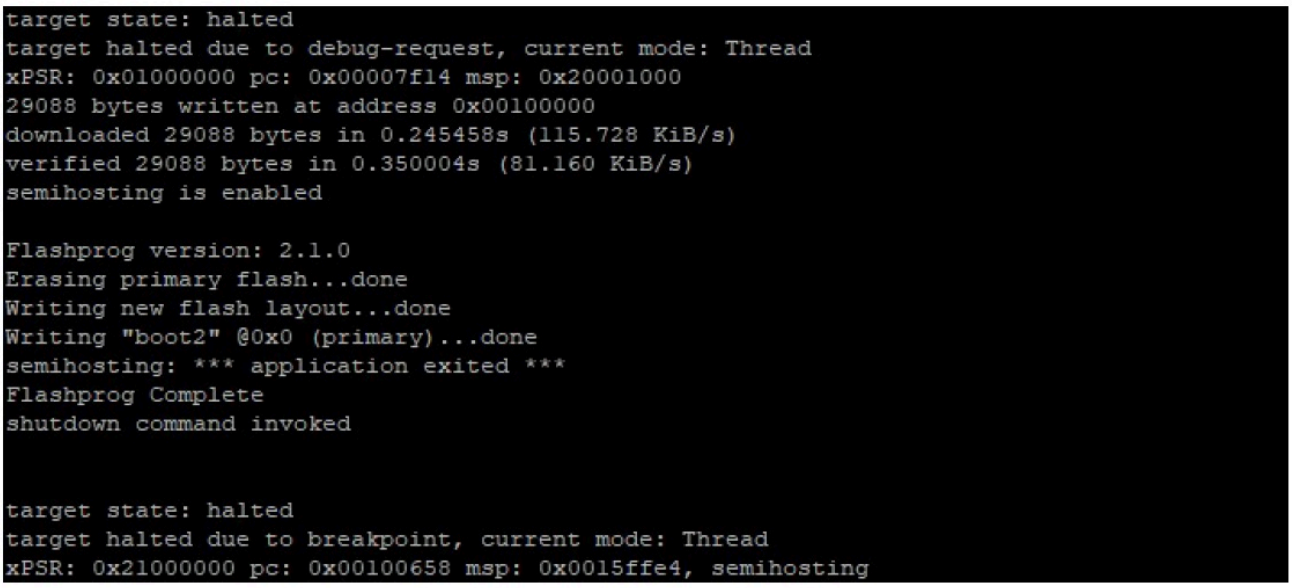
   a. Antes de instalar a imagem do firmware, prepare a instalação da placa de desenvolvimento com os componentes comuns, Layout e Boot2. Use os seguintes comandos.

   ```
   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
   ```

   O comando `flashprog` inicia o seguinte:

   - Layout: o utilitário flashprog é instruído primeiro a gravar um layout na instalação. O layout é semelhante às informações de partição da instalação. O layout padrão está localizado em `/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.

- Boot2: este é o carregador de inicialização usado pelo WMSDK. O flashprog também grava um carregador de inicialização na instalação. O carregador de inicialização carrega a imagem do firmware do microcontrolador depois que ela é instalada na placa. Obtenha a mesma saída mostrada na figura abaixo.



b. O firmware usa o chipset Wi-Fi para sua funcionalidade, e o chipset Wi-Fi tem seu próprio firmware que também deve estar presente na instalação. Você usa o utilitário `flashprog.py` para atualizar o firmware Wi-Fi da mesma forma que fez para atualizar o carregador de inicialização boot2 e o firmware do MCU. Use os comandos a seguir para instalar o firmware do 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
```

A saída do comando deve ser semelhante à figura abaixo.

```
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. Use os comandos a seguir para instalar o firmware da MCU.

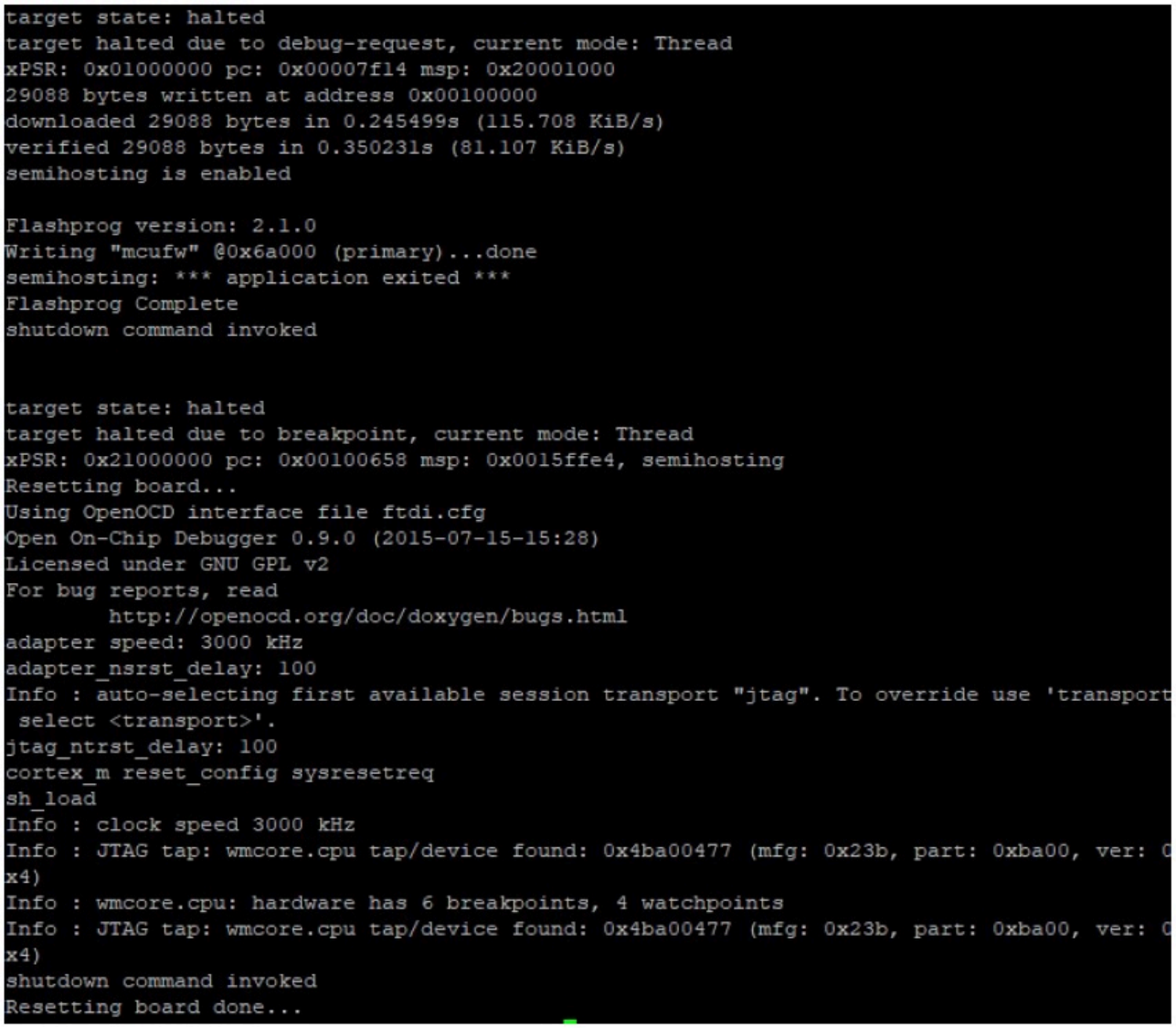
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. Redefina a placa. Você deve ver logs para a aplicação de demonstração.

e. Para executar a aplicação de teste, atualize o binário `aws_tests.bin` localizado no mesmo diretório.

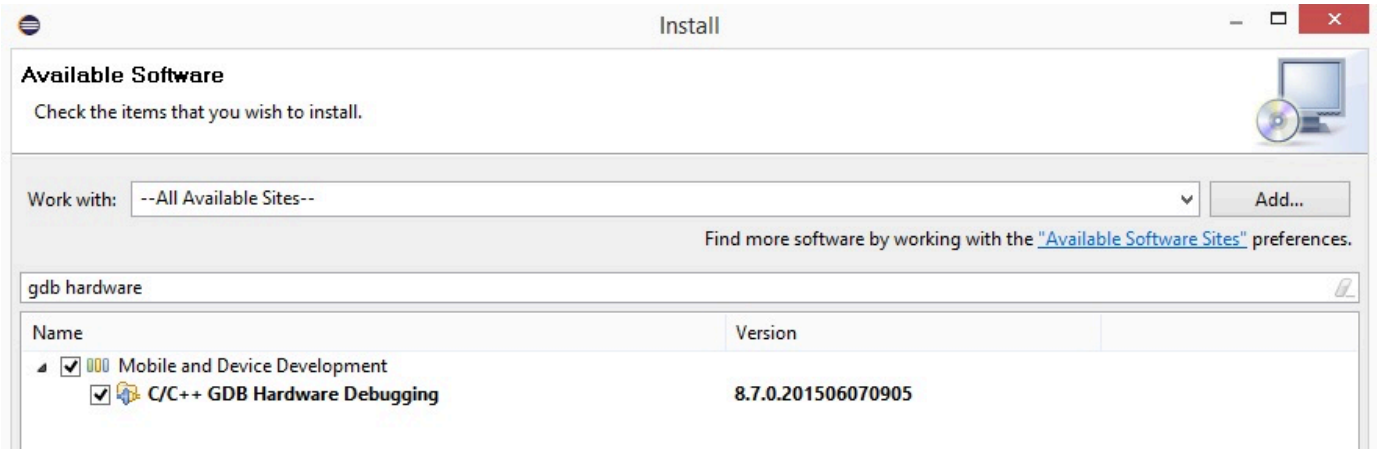
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

A saída de comando deve ser parecida com a mostrada na figura abaixo.



Depuração

- Inicie o Eclipse e escolha Ajuda, em seguida, escolha Instalar novo software. No menu Trabalhar com, escolha Todos os sites disponíveis. Insira o texto do filtro `GDB Hardware`. Selecione a opção Depuração de hardware GDB C/C++ e instale o plug-in.



Solução de problemas

Problemas de rede

Verifique suas credenciais de rede. Consulte "Provisionamento" em Compilação e execução do projeto de demonstração do FreeRTOS.

Habilitação de logs adicionais

- Habilitar logs específicos da placa.

  Habilite chamadas para `wmstdio_init(UART0_ID, 0)` na função `prvMiscInitialization` no arquivo `main.c` em testes ou demonstrações.

- Habilitação de logs de Wi-Fi

  Ative a macro `CONFIG_WLCMGR_DEBUG` no arquivo *`freertos`*`/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h`.

Uso do GDB

Recomendamos que você use os arquivos de comando `arm-none-eabi-gdb` e `gdb` empacotados junto com o SDK. Navegue até o diretório .

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

Execute o comando a seguir (em uma única linha) para se conectar ao GDB.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/
vendors/marvell/mw300 _rd/aws_demos.axf
```

Começando com o kit MediaTek MT7697Hx de desenvolvimento

> **Important**
>
> Essa integração de referência está hospedada no RTOS repositório Amazon-Free, que está obsoleto. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um RTOS projeto gratuito existente com base no repositório Amazon-FreeRTOS, agora obsoleto, consulte o. Guia de migração do repositório Github do Amazon FreeRTOS

Este tutorial fornece instruções para começar a usar o MediaTek MT7697Hx Development Kit. Se você não tiver o kit de MediaTek MT7697Hx desenvolvimento, visite o Catálogo de dispositivos do AWS parceiro para comprar um de nosso parceiro.

Antes de começar, você deve configurar AWS IoT e RTOS fazer o download gratuito para conectar seu dispositivo à AWS nuvem. Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de RTOS download gratuito é chamado de`freertos`.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilação cruzada de um aplicativo de RTOS demonstração gratuito em uma imagem binária.
3. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.
4. Interagir com o aplicativo em execução na placa em uma conexão serial para fins de monitoramento e depuração.

Configuração do ambiente de desenvolvimento

Antes de configurar seu ambiente, conecte seu computador à USB porta do MediaTek MT7697Hx Development Kit.

Baixe e instale Keil MDK

Você pode usar o Keil Microcontroller Development Kit (MDK) GUI baseado em Keil para configurar, criar e executar RTOS projetos gratuitos em sua placa. Keil MDK inclui o µVision IDE e o µVision Debugger.

> **Note**
>
> O Keil MDK é compatível somente com máquinas Windows 7, Windows 8 e Windows 10 de 64 bits.

Para baixar e instalar Keil MDK

1. Vá para a página de MDKintrodução do Keil e escolha Download MDK -Core.
2. Insira e envie suas informações para serem registradas com o Keil.
3. Clique com o botão direito do mouse no MDK executável e salve o MDK instalador do Keil em seu computador.
4. Abra o MDK instalador do Keil e siga as etapas até a conclusão. Certifique-se de instalar o pacote de MediaTek dispositivos (MT76x7Série).

Estabelecimento de uma conexão serial

Conecte a placa ao seu computador host com um USB cabo. Uma COM porta aparece no Gerenciador de dispositivos do Windows. Para depuração, você pode abrir uma sessão na porta com uma ferramenta utilitária de terminal, como ou. HyperTerminal TeraTerm

Monitorando MQTT mensagens na nuvem

Antes de executar o projeto de RTOS demonstração gratuito, você pode configurar o MQTT cliente no AWS IoT console para monitorar as mensagens que seu dispositivo envia para a AWS nuvem.

Para assinar o MQTT tópico com o AWS IoT MQTT cliente

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Testar e, em seguida, escolha cliente MQTT de teste para abrir o MQTT cliente.
3. Em Tópico de inscrição, insira ***your-thing-name*****example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Crie e execute o projeto de RTOS demonstração gratuito com Keil MDK

Para criar o projeto de RTOS demonstração gratuito em Keil µVision

1. No menu Iniciar, abra o Keil µVision 5.
2. Abra o arquivo de projeto `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx`.
3. No menu, escolha Project (Projeto) e Build target (Compilar destino).

   Depois que o código for compilado, você verá o arquivo executável de demonstração em `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf`.

Para executar o projeto de RTOS demonstração gratuito

1. Coloque o kit MediaTek MT7697Hx de desenvolvimento PROGRAM no modo.

   Para colocar o kit PROGRAM no modo, pressione e segure o PROGbotão. Com o PROGbotão ainda pressionado, pressione e solte o RESETbotão e, em seguida, solte-o PROG.
2. No menu, escolha Flash e Configure Flash Tools (Configurar ferramentas de atualização).
3. Em Options for Target (Opções de destino) "**`aws_demo`**", escolha a guia Debug (Depurar). Selecione Usar, defina o depurador como CMSIS- DAP Depurador e escolha OK.
4. No menu, escolha Flash (Atualizar) e Download (Fazer download).

   O µVision notificará você quando o download for concluído.
5. Use um utilitário de terminal para abrir a janela do console de série. Defina a porta serial como 115200 bps, sem paridade, 8 bits e 1 bit de parada.
6. Escolha o RESETbotão no seu kit MediaTek MT7697Hx de desenvolvimento.

Solução de problemas

Depuração de RTOS projetos gratuitos em Keil µVision

Atualmente, você deve editar o MediaTek pacote incluído no Keil µVision antes de poder depurar o projeto de RTOS demonstração gratuito MediaTek com Keil µVision.

Para editar o MediaTek pacote para depuração de projetos gratuitos RTOS

1. Encontre e abra o `Keil_v5\ARM\PACK\.Web\MediaTek.MTx.pdsc` arquivo na pasta de MDK instalação do Keil.
2. Substitua todas as instâncias de `flag = Read32(0x20000000);` por `flag = Read32(0x0010FBFC);`.
3. Substitua todas as instâncias de `Write32(0x20000000, 0x76877697);` por `Write32(0x0010FBFC, 0x76877697);`.

Para iniciar a depuração do projeto

1. No menu, escolha Flash e Configure Flash Tools (Configurar ferramentas de atualização).
2. Escolha a guia Target (Destino) e, depois, Read/Write Memory Areas (Ler/gravar áreas de memória). Confirme isso IRAM1 e ambos IRAM2 estão selecionados.
3. Escolha a guia Depurar e, em seguida, escolha CMSIS- DAP Depurador.
4. Abra `vendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c` e defina a macro `MTK_DEBUGGER` como `1`.
5. Compile novamente o projeto de demonstração no µVision.
6. Coloque o kit MediaTek MT7697Hx de desenvolvimento PROGRAM no modo.

   Para colocar o kit PROGRAM no modo, pressione e segure o PROGbotão. Com o PROGbotão ainda pressionado, pressione e solte o RESETbotão e, em seguida, solte-o PROG.
7. No menu, escolha Flash (Atualizar) e Download (Fazer download).

   O µVision notificará você quando o download for concluído.
8. Pressione o RESETbotão no seu kit MediaTek MT7697Hx de desenvolvimento.
9. No menu do µVision, escolha Depuração e Iniciar/Interromper sessão de depuração. A janela Chamar pilha + locais é aberta quando você inicia a sessão de depuração.
10. No menu, escolha Debug (Depurar) e, depois, escolha Stop (Parar) para pausar a execução do código. O contador do programa é interrompido na seguinte linha:

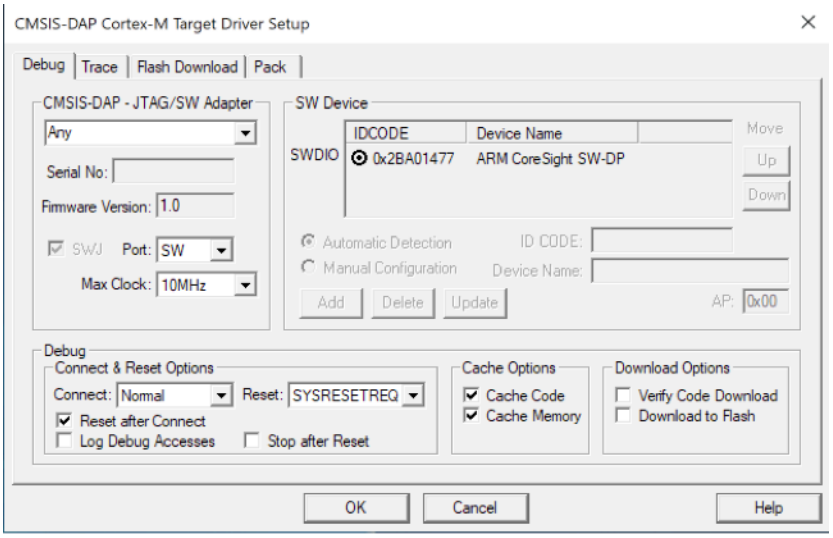
    ```
    { volatile int wait_ice = 1 ; while ( wait_ice ) ; }
    ```
11. Na janela Call Stack + Locals (Chamar pilha + locais), altere o valor de `wait_ice` para `0`.
12. Defina os pontos de interrupção no código-fonte do projeto e execute-o.

Solução de problemas nas configurações do IDE depurador

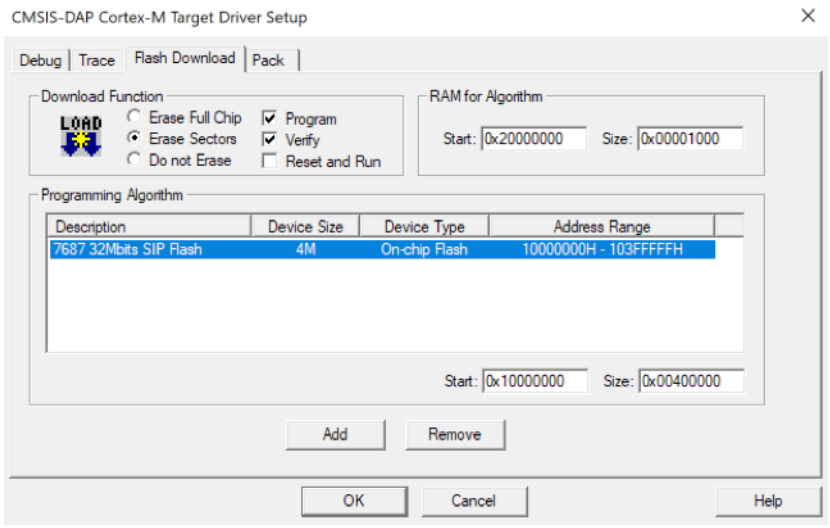
Se você está com problemas para depurar uma aplicação, as configurações do seu depurador podem estar incorretas.

Para verificar se as configurações do depurador estão corretas

1. Abra o Keil µVision.
2. Clique com o botão direito do mouse no projeto `aws_demos`, escolha Options (Opções), e na guia Utilities (Utilitários), escolha Settings (Configurações) ao lado de “-- Use Debug Driver --” (Usar driver de depuração).
3. Verifique se as configurações na guia Debug (Depurar) são exibidas da seguinte forma:



4. Verifique se as configurações na guia Flash Download (Atualizar download) são exibidas da seguinte forma:



Para obter informações gerais sobre solução de problemas sobre como começar a usar o FreeRTOS, consulteSolução de problemas de conceitos básicos.

Conceitos básicos do Microchip Curiosity PIC32MZ EF

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

> **Note**
>
> De acordo com a Microchip, estamos removendo o Curiosity PIC32MZEF (DM320104) da ramificação principal do repositório de Integração de Referência do FreeRTOS e não o carregaremos mais em novas versões. A Microchip emitiu um aviso oficial de que o PIC32MZEF (DM320104) não é mais recomendado para novos designs. Os projetos e o código-fonte do PIC32MZEF ainda podem ser acessados por meio das tags de lançamento anteriores. A Microchip recomenda que os clientes usem a placa de desenvolvimento Curiosity PIC32MZ-EF-2.0 (DM320209) para novos designs. A plataforma PIC32MZv1 ainda pode ser encontrada na versão v202012.00 do repositório de Integração de Referência do FreeRTOS. No entanto, não há mais suporte para a plataforma não na versão v202107.00 da Referência do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Microchip Curiosity PIC32MZ EF. Se você não tiver o pacote Microchip Curiosity PIC32MZ EF, acesse o catálogo do AWS Partner Device para adquirir um dos nossos parceiros.

O pacote inclui os seguintes itens:

- Placa de desenvolvimento Curiosity PIC32MZ EF
- Placa MikroElectronika USB UART Click
- Placa MikroElectronika WiFi 7 Click
- Placa auxiliar PIC32 LAN8720 PHY

Você também precisa os seguintes itens para depuração:

- Depurador MPLAB Snap In-Circuit
- (Opcional) Kit de cabos de programação PICkit 3

Antes de começar, você deve configurar o AWS IoT e seu download do FreeRTOS para conectar seu dispositivo à nuvem da AWS. Para obter instruções, consulte Primeiras etapas.

Important

- Neste tópico, o caminho para o diretório de download do FreeRTOS é chamado de *`freertos`*.
- Caracteres de espaço no caminho *`freertos`* podem causar falhas na compilação. Ao clonar ou copiar o repositório, verifique se o caminho criado não contém caracteres de espaço.
- O tamanho máximo de um caminho de arquivo no Microsoft Windows é 260 caracteres. Caminhos longos de diretório de download do FreeRTOS podem causar falhas de compilação.
- O código-fonte pode conter links simbólicos, por isso se estiver usando o Windows para extrair o arquivo, talvez seja necessário:
  - Habilitar o modo Desenvolvedor ou,
  - Usar um console com privilégios de administrador.

  Dessa forma, o Windows pode criar links simbólicos adequadamente ao extrair o arquivamento. Caso contrário, os links simbólicos serão gravados como arquivos comuns que contêm os caminhos dos links simbólicos como texto ou estão vazios. Para obter mais informações, consulte a entrada no blog Symlinks no Windows 10!.

  Se você usa o Git no Windows, você deve habilitar o modo Desenvolvedor ou deve:
  - Definir `core.symlinks` como verdadeiro com o seguinte comando:

    ```
    git config --global core.symlinks true
    ```
  - Use um console com privilégios de administrador sempre que usar um comando git que grava no sistema (por exemplo, git pull, git clone, e git submodule update --init --recursive).

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Conectar sua placa a uma máquina host.
2. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
3. Compilar um aplicativo de demonstração do FreeRTOS de forma cruzada para uma imagem binária.
4. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.
5. Interagir com o aplicativo em execução na placa em uma conexão serial para fins de monitoramento e depuração.

Configuração do hardware Microchip Curiosity PIC32MZ EF

1. Conecte a placa MikroElectronika USB UART click ao conector microBUS 1 no Microchip Curiosity PIC32MZ EF.
2. Conecte a placa auxiliar PIC32 LAN8720 PHY ao cabeçalho J18 no Microchip Curiosity PIC32MZ EF.
3. Conecte a placa MikroElectronika USB UART click ao computador usando um cabo USB A para USB mini-B.
4. Para conectar a placa à Internet, use uma das seguintes opções:
   - Para usar Wi-Fi, conecte a placa de clique MikroElectronika Wi-Fi 7 ao conector microBUS 2 no Microchip Curiosity PIC32MZ EF. Consulte Configuração das demonstrações do FreeRTOS.
   - Para usar Ethernet para conectar a placa Microchip Curiosity PIC32MZ EF à internet, conecte a placa filha PIC32 LAN8720 PHY ao cabeçalho J18 no Microchip Curiosity PIC32MZ EF. Conecte uma extremidade do cabo Ethernet à placa auxiliar LAN8720 PHY. Conecte a outra extremidade ao roteador ou outra porta da Internet. Você também deve definir a macro `PIC32_USE_ETHERNET` do pré-processador.
5. Se ainda não tiver feito isso, solde o conector angular ao conjunto ICSP no Microchip Curiosity PIC32MZ EF.
6. Conecte uma extremidade do cabo ICSP do Kit de cabos de programação PICkit 3 ao Microchip Curiosity PIC32MZ EF.

Se não tiver Kit de cabos de programação PICkit 3, você poderá usar jumpers de cabo M-F Dupont para estabelecer a conexão. Observe que o círculo branco indica a posição do Pino 1.

7. Conecte a outra extremidade do cabo ICSP (ou jumpers) ao Depurador MPLAB Snap. O Pino 1 do Conector de programação SIL de 8 pinos está marcado com um triângulo preto na parte inferior direita da placa.

   Certifique-se de que qualquer cabo conectado ao Pino 1 no Microchip Curiosity PIC32MZ EF, indicado pelo círculo branco, esteja alinhado ao Pino 1 no Depurador MPLAB Snap.

   Para obter mais informações sobre o Depurador MPLAB Snap, consulte MPLAB Snap In-Circuit Debugger Information Sheet.

Configuração do hardware do Microchip Curiosity PIC32MZ EF usando PICkit On Board (PKOB)

Recomendamos seguir o procedimento de configuração na seção anterior. No entanto, você pode avaliar e executar demonstrações do FreeRTOS com depuração básica usando o programador/depurador do PICkit On Board (PKOB) integrado seguindo estas etapas.

1. Conecte a placa MikroElectronika USB UART click ao conector microBUS 1 no Microchip Curiosity PIC32MZ EF.
2. Para conectar a placa à Internet, siga um destes procedimentos:

   - Para usar Wi-Fi, conecte a placa de clique MikroElectronika Wi-Fi 7 ao conector microBUS 2 no Microchip Curiosity PIC32MZ EF. (Siga as etapas em “Como configurar seu Wi-Fi” em Configuração das demonstrações do FreeRTOS).
   - Para usar Ethernet para conectar a placa Microchip Curiosity PIC32MZ EF à internet, conecte a placa filha PIC32 LAN8720 PHY ao cabeçalho J18 no Microchip Curiosity PIC32MZ EF. Conecte uma extremidade do cabo Ethernet à placa auxiliar LAN8720 PHY. Conecte a outra extremidade ao roteador ou outra porta da Internet. Você também deve definir a macro `PIC32_USE_ETHERNET` do pré-processador.
3. Conecte a porta USB micro-B chamada “USB DEBUG” na placa Microchip Curiosity PIC32MZ EF ao seu computador usando um cabo USB tipo A para USB micro-B.
4. Conecte a placa MikroElectronika USB UART click ao computador usando um cabo USB A para USB mini-B.

Configuração do ambiente de desenvolvimento

> **Note**
>
> O projeto do FreeRTOS para esse dispositivo é baseado no MPLAB Harmony v2. Para criar o projeto, você precisa usar versões das ferramentas do MPLAB que são compatíveis com Harmony v2, como v2.10 do MPLAB XC32 Compiler e as versões 2.X.X do MPLAB Harmony Configurator (MHC).

1. Instale o Python versão 3.x ou posterior.
2. Instale o MPLAB X IDE:

   > **Note**
   >
   > Atualmente, há suporte para as Integrações de referência v202007.00 do AWS FreeRTOS somente no MPLabv5.35. Há suporte para as versões anteriores das Integrações de referência do AWS FreeRTOS no MPLabv5.40.

   Downloads de MPLab v5.35

   - MPLAB X Integrated Development Environment para Windows
   - MPLAB X Integrated Development Environment para macOS
   - MPLAB X Integrated Development Environment para Linux

   Últimos downloads do MPlab (MPLab v5.40)

   - MPLAB X Integrated Development Environment para Windows
   - MPLAB X Integrated Development Environment para macOS
   - MPLAB X Integrated Development Environment para Linux
3. Instale o MPLAB XC32 Compiler:
   - MPLAB XC32/32++ Compiler para Windows
   - MPLAB XC32/32++ Compiler para macOS
   - MPLAB XC32/32++ Compiler para Linux

4. Inicie um emulador de terminal UART e abra uma conexão com as seguintes configurações:

   - Taxa de baud: 115200
   - Dados: 8 bits
   - Paridade: nenhum
   - Bits de parada: 1
   - Controle de fluxo: nenhum

Monitoramento de mensagens MQTT na nuvem

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console do AWS IoT para monitorar as mensagens enviadas pelo dispositivo para a nuvem da AWS.

Para assinar o tópico MQTT com o cliente MQTT do AWS IoT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Compilação e execução do projeto de demonstração do FreeRTOS

Abrir a demonstração do FreeRTOS no IDE do MPLAB

1. Abra o MPLAB IDE. Se você tiver mais de uma versão do compilador instalado, você precisa selecionar o compilador que você deseja usar de dentro do IDE.
2. No menu File (Arquivo), escolha Open project (Abrir projeto).
3. Navegue e abre `projects/microchip/curiosity_pic32mzef/mplab/aws_demos`.
4. Selecione Open project (Abrir projeto).

> **Note**
>
> Quando você abre o projeto pela primeira vez, pode aparecer uma mensagem de erro sobre o compilador. No IDE, navegue até Tools (Ferramentas), Options (Opções), Embedded (Incorporado) e selecione o compilador que você está usando para seu projeto.

Para usar a Ethernet para se conectar, você deve definir a macro do pré-processador `PIC32_USE_ETHERNET`.

Como usar Ethernet para se conectar usando o IDE do MPLAB

1. No IDE do MPLAB, clique com o botão direito do mouse no projeto e escolha Propriedades.
2. Na caixa de diálogo Propriedades do projeto, escolha ***nome do compilador*** (Opções globais) para expandi-lo e selecione nome do ***compilador***-gcc.
3. Em categorias de Opções, escolha Pré-processamento e mensagens e, em seguida, adicione a string `PIC32_USE_ETHERNET` às macros do pré-processador.

Execução do projeto de demonstração do FreeRTOS

1. Compile o projeto novamente.
2. Na guia Projects (Projetos), clique com o botão direito do mouse na pasta de nível superior `aws_demos` e, em seguida, escolha Debug (Depurar).
3. Quando o depurador parar no ponto de interrupção em `main()`, no menu Run (Executar), escolha Resume (Continuar).

Compilação da demonstração do FreeRTOS com CMake

Se você preferir não usar um IDE para desenvolvimento do FreeRTOS, também é possível usar o CMake para compilar e executar os aplicativos de demonstração ou aplicativos que você desenvolveu usando editores de código e ferramentas de depuração de terceiros.

Como compilar a demonstração do FreeRTOS com CMake

1. Crie um diretório para a contenção dos arquivos de compilação gerados, como *build-directory*.
2. Use o comando a seguir para gerar arquivos de compilação do código-fonte.

```
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**
>
> Você deve especificar os caminhos corretos para os binários da cadeia de ferramentas e do Hexmate, como os caminhos `C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform\bin` e `C:\Program Files\Microchip\xc32\v2.40\bin`.

3. Altere os diretórios para o diretório de compilação (*build-directory*) e, em seguida, execute `make` a partir desse diretório.

Para obter mais informações, consulte Uso da CMake com o FreeRTOS.

Para usar a Ethernet para se conectar, você deve definir a macro do pré-processador `PIC32_USE_ETHERNET`.

Solução de problemas

Para obter informações sobre a solução de problemas, consulte Solução de problemas de conceitos básicos.

Conceitos básicos do Nordic nRF52840-DK

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar o Nordic nRF52840-DK. Se você não tiver o Nordic nRF52840-DK, acesse o catálogo do AWS Partner Device para comprar um com nossos parceiros.

Antes de começar, você precisa Configure o AWS IoT Amazon Cognito para Freertos Bluetooth Low Energy.

Para executar a demonstração de Bluetooth Low Energy do FreeRTOS, você também precisa de um dispositivo móvel Android ou iOS com recursos de Bluetooth e Wi-Fi.

> **Note**
>
> Se você estiver usando um dispositivo iOS, precisará do Xcode para criar o aplicativo móvel de demonstração. Se você estiver usando um dispositivo Android, poderá usar o Android Studio para criar o aplicativo móvel de demonstração.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Conectar sua placa a uma máquina host.
2. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
3. Compilar um aplicativo de demonstração do FreeRTOS de forma cruzada para uma imagem binária.
4. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.
5. Interagir com o aplicativo em execução na placa em uma conexão serial para fins de monitoramento e depuração.

Configuração do hardware Nordic

Conecte seu computador host à porta USB chamada de J2, localizada diretamente acima do compartimento da bateria de célula em forma de moeda na sua placa Nordic nRF52840.

Para obter mais informações sobre como configurar o Nordic nRF52840-DK, consulte o Guia do usuário do kit de desenvolvimento do nRF52840.

Configuração do ambiente de desenvolvimento

Download e instalação do Segger Embedded Studio

O FreeRTOS oferece suporte ao Segger Embedded Studio como um ambiente de desenvolvimento para o Nordic nRF52840-DK.

Para configurar seu ambiente, você precisa fazer download do Segger Embedded Studio e instalá-lo no seu computador host.

Para fazer download do Segger Embedded Studio e instalá-lo

1. Vá para a página Downloads do Segger Embedded Studio e escolha a opção Embedded Studio para ARM do seu sistema operacional.
2. Execute o instalador e siga as instruções para a conclusão.

Configuração do aplicativo de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS

Para executar o projeto de demonstração do FreeRTOS no Bluetooth Low Energy, é necessário executar o aplicativo de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS em um dispositivo móvel.

Como configurar o aplicativo de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS

1. Siga as instruções em SDKs móveis para dispositivos Bluetooth do FreeRTOS para fazer download e instalar o SDK para sua plataforma móvel em seu computador host.
2. Siga as instruções em aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS para configurar o aplicativo móvel de demonstração no dispositivo móvel.

Estabelecimento de uma conexão serial

O Segger Embedded Studio inclui um emulador de terminal que você pode usar para receber mensagens de log em uma conexão serial com sua placa.

Para estabelecer uma conexão serial com o Segger Embedded Studio

1. Abra o Segger Embedded Studio.
2. No menu superior, escolha Target (Destino), Connect J-Link (Conectar J-Link).
3. No menu superior, escolha Tools (Ferramentas), Terminal Emulator (Emulador de terminal), Properties (Propriedades) e defina as propriedades de acordo com as instruções em Instalação de um emulador de terminal.
4. No menu superior, escolha Tools (Ferramentas), Terminal Emulator (Emulador de terminal), Connect ***port*** (Porta Connect) (115200,N,8,1).

> **Note**
>
> O emulador de terminal de estúdio incorporado Segger não oferece suporte a uma capacidade de entrada. Para isso, use um emulador de terminal como PuTTy, Tera Term ou GNU Screen. Configure o terminal para conectar-se à sua placa por uma conexão serial de acordo com as instruções em Instalação de um emulador de terminal.

Faça download e configure os FreeRTOS

Depois de configurar o hardware e o ambiente, faça download do FreeRTOS.

Baixe o FreeRTOS

Para fazer download do FreeRTOS para o Nordic nRF52840-DK, acesse a página GitHub do FreeRTOS e clone o repositório. Consulte o arquivo README.md para obter instruções.

> **Important**
>
> - Neste tópico, o caminho para o diretório de download do FreeRTOS é chamado de *`freertos`*.
> - Caracteres de espaço no caminho *`freertos`* podem causar falhas na compilação. Ao clonar ou copiar o repositório, verifique se o caminho criado não contém caracteres de espaço.
> - O tamanho máximo de um caminho de arquivo no Microsoft Windows é 260 caracteres. Caminhos longos de diretório de download do FreeRTOS podem causar falhas de compilação.
> - O código-fonte pode conter links simbólicos, por isso se estiver usando o Windows para extrair o arquivo, talvez seja necessário:
>   - Habilitar o modo Desenvolvedor ou,
>   - Usar um console com privilégios de administrador.
>
>   Dessa forma, o Windows pode criar links simbólicos adequadamente ao extrair o arquivamento. Caso contrário, os links simbólicos serão gravados como arquivos comuns que contêm os caminhos dos links simbólicos como texto ou estão vazios. Para obter mais informações, consulte a entrada no blog Symlinks no Windows 10!.
>
>   Se você usa o Git no Windows, você deve habilitar o modo Desenvolvedor ou deve:

- Definir `core.symlinks` como verdadeiro com o seguinte comando:

```
git config --global core.symlinks true
```

- Use um console com privilégios de administrador sempre que usar um comando git que grava no sistema (por exemplo, git pull, git clone, e git submodule update --init --recursive).

Configuração do projeto

Para executar a demonstração, você deve configurar seu projeto para funcionar com o AWS IoT. Para configurar seu projeto para funcionar com a AWS IoT, o dispositivo deve estar registrado como uma coisa da AWS IoT. Você deve ter registrado o dispositivo quando Configure o AWS IoT Amazon Cognito para Freertos Bluetooth Low Energy.

Para configurar o endpoint da AWS IoT

1. Faça login no console do AWS IoT.
2. No painel de navegação, selecione Settings (configurações).

   O endpoint da AWS IoT é exibido na caixa de texto Endpoint de dados de dispositivo. Deve ser semelhante a *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Anote esse endpoint.
3. No painel de navegação, escolha Manage (Gerenciar) e, depois, Things (Coisas). Anote o nome da coisa da AWS IoT para o seu dispositivo.
4. Com o endpoint da AWS IoT e o nome da coisa da AWS IoT disponíveis, abra *freertos*`/demos/include/aws_clientcredential.h` no IDE e especifique os valores das seguintes constantes `#define`:

   - `clientcredentialMQTT_BROKER_ENDPOINT` *Seu endpoint da AWS IoT*
   - `clientcredentialIOT_THING_NAME` *O nome da coisa da AWS IoT da sua placa*

Para habilitar a demonstração

1. Verifique se a demonstração Bluetooth Low Energy GATT está habilitada. Navegue até `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/`

`iot_ble_config.h` e adicione `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` à lista de instruções define.

2. Abra `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h` e defina um `CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED` como neste exemplo.

```
/* 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. Como o chip Nordic é fornecido com muito pouca RAM (250 KB), a configuração de BLE pode precisar ser alterada para permitir entradas de tabela GATT maiores em comparação com o tamanho de cada atributo. Dessa forma, você pode ajustar a quantidade de memória que o aplicativo recebe. Para fazer isso, substitua as definições dos seguintes atributos no arquivo *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h`:

   - NRF_SDH_BLE_VS_UUID_COUNT

     O número de UUIDs específicos do fornecedor. Aumente essa contagem em 1 ao adicionar um novo UUID específico do fornecedor.

   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

     O tamanho da tabela de atributos em bytes. O tamanho deve ser um múltiplo de 4. Esse valor indica a quantidade definida de memória dedicada para a tabela de atributos (incluindo o tamanho da característica), portanto, isso variará de projeto para projeto. Se você exceder o tamanho da tabela de atributos, receberá um erro NRF_ERROR_NO_MEM. Se você modificar o NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE, normalmente também deverá redefinir as configurações de RAM.

(Para testes, a localização do arquivo é *freertos*/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h.)

Compilação e execução do projeto de demonstração do FreeRTOS

Depois de fazer download do FreeRTOS e configurar o projeto de demonstração, você poderá compilar e executar o projeto de demonstração na placa.

> **Important**
>
> Se esta for a primeira vez que você está executando a demonstração nesta placa, você precisa atualizar uma bootloader à placa antes da execução da demonstração. Para criar e atualizar a bootloader, siga as etapas abaixo, mas, em vez de usar o arquivo de projeto `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`, use `projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject`.

Como compilar e executar a demonstração de Bluetooth Low Energy do FreeRTOS no Segger Embedded Studio

1. Abra o Segger Embedded Studio. No menu superior, escolha File (Arquivo), Open Solution (Abrir solução) e navegue até o arquivo do projeto `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`
2. Se você estiver usando o emulador de terminal Segger Embedded Studio, escolha Tools (Ferramentas) no menu superior e, depois, Terminal Emulator (Emulador de terminal), Terminal Emulator (Emulador de terminal) para exibir informações da sua conexão serial.

   Se você usa outra ferramenta de terminal, pode monitorar a saída dessa ferramenta a partir da sua conexão serial.
3. Clique com o botão direito no projeto de demonstração `aws_demos` no Project Explorer (Explorador de projeto) e escolha Build (Compilar).

> **Note**
>
> Se esta é a primeira vez que você usa o Segger Embedded Studio, é possível que o aviso "Nenhuma licença para uso comercial" seja exibido. O Segger Embedded Studio pode ser usado gratuitamente para dispositivos semicondutores Nordic. Solicite uma licença gratuita e, durante a configuração, escolha Ativar a licença gratuita e siga as instruções.

4. Escolha Debug (Depurar) e, depois, Go (Ir).

   Após a demonstração ser iniciada, ela aguarda para emparelhar com um dispositivo móvel em Bluetooth Low Energy.

5. Siga as instruções do Aplicativo de demonstração do MQTT por Bluetooth Low Energy para concluir a demonstração com o aplicativo de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS como o proxy MQTT móvel.

Solução de problemas

Para obter mais informações sobre soluções de problemas gerais sobre os Conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Introdução ao NuMaker Nuvoton -IoT-M487

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar a placa de desenvolvimento Nuvoton NuMaker -IoT-M487. O microcontrolador de série inclui módulos RJ45 Ethernet e Wi-Fi integrados. Se você não tiver o Nuvoton NuMaker -IoT-M487, visite o Catálogo de dispositivos do parceiro para comprar um de AWS nosso parceiro.

Antes de começar, você deve configurar AWS IoT seu software FreeRTOS para conectar sua placa de desenvolvimento à nuvem. AWS Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de download do FreeRTOS é chamado *`freertos`*.

Visão geral

Este tutorial orienta você pelas seguintes etapas:

1. Instalar software na máquina host para desenvolver e depurar aplicações incorporadas para a placa do microcontrolador.
2. Fazer a compilação cruzada de uma aplicação de demonstração do FreeRTOS para uma imagem binária.
3. Carregar a imagem binária do aplicativo em na placa e executar o aplicativo.

Configuração do ambiente de desenvolvimento

A edição Keil MDK Nuvoton foi projetada para desenvolver e depurar aplicações para placas Nuvoton M487. A versão Essential, Plus ou Pro do Keil MDK v5 também deve funcionar para o MCU Nuvoton M487 (núcleo Cortex-M4). Você pode fazer download da edição Keil MDK Nuvoton com um desconto para os MCUs da série Nuvoton Cortex-M4. O Keil MDK só é compatível com o Windows.

Para instalar a ferramenta de desenvolvimento para o NuMaker -IoT-M487

1. Faça download da Edição Keil MDK Nuvoton no site do Keil MDK.
2. Instale o Keil MDK em sua máquina host usando sua licença. O Keil MDK inclui o Keil µVision IDE, uma cadeia de ferramentas de compilação C/C++ e o depurador µVision.

   Se você tiver problemas durante a instalação, entre em contato com a Nuvoton para obter assistência.
3. Instale oNu-Link_Keil_Driver_V3.06.7215r (ou versão mais recente), que está na página Ferramenta de desenvolvimento do Nuvoton.

Compilação e execução do projeto de demonstração do FreeRTOS

Como compilar o projeto de demonstração do FreeRTOS

1. Abra o IDE Keil µVision.

2. No menu File (Arquivo), escolha Open (Abrir). Na caixa de diálogo Open file (Abrir arquivo), verifique se o seletor de tipo de arquivo está definido como Project Files (Arquivos de projeto).
3. Escolha o projeto de demonstração Wi-Fi ou Ethernet a ser criado.
   - Para abrir o projeto de demonstração Wi-Fi, escolha o projeto de destino `aws_demos.uvproj` no diretório *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos`.
   - Para abrir o projeto de demonstração Ethernet, escolha o projeto de destino `aws_demos_eth.uvproj` no diretório *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth`.
4. Para garantir que suas configurações estejam corretas para atualizar a placa, clique com o botão direito do mouse no projeto `aws_demo` no IDE e escolha Options (Opções). (Consulte Solução de problemas para obter mais detalhes.)
5. Na guia Utilities (Utilitários) verifique se Use Target Driver for Flash Programming (Usar o driver de destino para programação flash) está selecionado e se Nuvoton Nu-Link Debugger (Depurador Nuvoton Nu-Link) está definido como o driver de destino.
6. Na guia Debug (Depurar), ao lado de Nuvoton Nu-Link Debugger (Depurador Nuvoton Nu-Link), escolha Settings (Configurações).
7. Verifique se Chip Type (Tipo de chip) está definido como M480.
8. No painel de navegação Project (Projeto) do IDE do Keil µVision, escolha o projeto `aws_demos`. No menu Project (Projeto), escolha Build Target (Compilar destino).

Você pode usar o cliente MQTT no AWS IoT console para monitorar as mensagens que seu dispositivo envia para a AWS nuvem.

Para assinar o tópico MQTT com o cliente AWS IoT MQTT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***`your-thing-name`*`/example/topic`** e selecione Inscreva-se no tópico.

Como executar o projeto de demonstração do FreeRTOS

1. Conecte a placa Numaker-IoT-M487 à máquina host (computador).
2. Recrie o projeto.
3. No Keil µVision IDE, no menu Flash (Atualizar) escolha Download (Fazer download).
4. No menu Debug (Depurar), escolha Start/Stop Debug Session (Iniciar/interromper sessão de depuração).
5. Quando o depurador parar no ponto de interrupção em `main()`, abra o menu Run (Executar) e escolha Run (F5) (Executar (F5)).

   Você deve ver as mensagens MQTT enviadas pelo seu dispositivo no cliente MQTT no AWS IoT console.

Uso da CMake com o FreeRTOS

Você também pode usar o CMake para compilar e executar aplicações de demonstração do FreeRTOS ou aplicações que você desenvolveu usando editores de código e ferramentas de depuração de terceiros.

Verifique se você instalou o sistema de compilação do CMake. Siga as instruções em Uso da CMake com o FreeRTOS e depois siga as etapas nesta seção.

> **Note**
>
> Certifique-se de que o caminho para o local do compilador (Keil) esteja na variável do sistema Path, por exemplo, `C:\Keil_v5\ARM\ARMCC\bin`.

Você também pode usar o cliente MQTT no AWS IoT console para monitorar as mensagens que seu dispositivo envia para a AWS nuvem.

Para assinar o tópico MQTT com o cliente AWS IoT MQTT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*/example/topic** e selecione Inscreva-se no tópico.

Para gerar arquivos de compilação a partir de arquivos de origem e executar o projeto de demonstração

1. Na máquina host, abra o prompt de comando e navegue até a pasta *freertos*.
2. Crie uma pasta para conter os arquivos de compilação gerados. Chamaremos essa pasta de *BUILD_FOLDER*.
3. Gere os arquivos de compilação para a demonstração Wi-Fi ou Ethernet.

   - Para Wi-Fi:

     Navegue até o diretório que contém os arquivos de origem do projeto de demonstração do FreeRTOS. Depois, gere os arquivos de compilação executando o seguinte comando.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -
     B BUILD_FOLDER -G Ninja
     ```

   - Para Ethernet:

     Navegue até o diretório que contém os arquivos de origem do projeto de demonstração do FreeRTOS. Depois, gere os arquivos de compilação executando o seguinte comando.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -
     DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
     ```

4. Gere o binário para refletir no M487 executando o seguinte comando.

   ```
   cmake --build BUILD_FOLDER
   ```

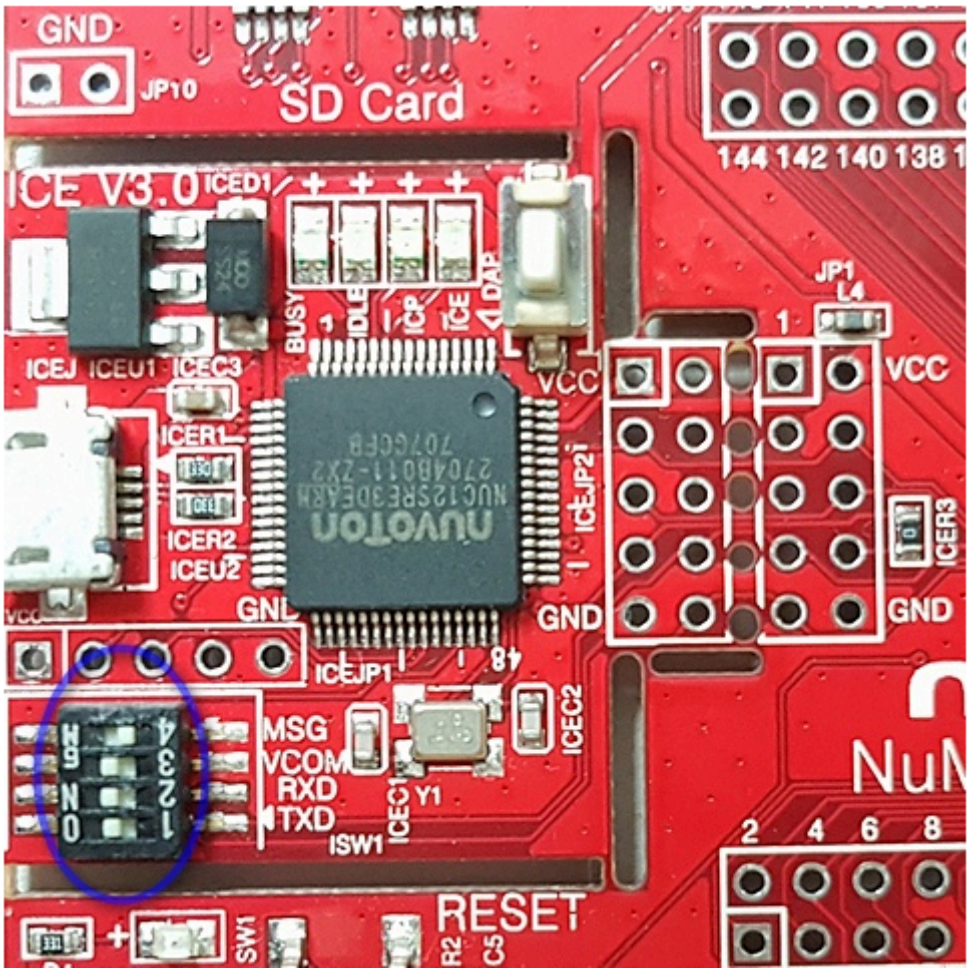
   Nesse ponto, o arquivo binário `aws_demos.bin` deve estar na pasta *BUILD_FOLDER*`/vendors/Nuvoton/boards/numaker_iot_m487_wifi`.

5. Para configurar a placa para o modo de atualização, certifique-se de que o switch MSG (No.4 do ISW1 no ICE) esteja ativado. Quando você conecta a placa, uma janela (e unidade) será atribuída. (Consulte Solução de problemas.)
6. Abra um emulador de terminal para visualizar as mensagens via UART. Siga as instruções em Instalação de um emulador de terminal.
7. Execute o projeto de demonstração copiando o binário gerado no dispositivo.

Se você se inscreveu no tópico MQTT com o cliente AWS IoT MQTT, deverá ver as mensagens MQTT enviadas pelo seu dispositivo no console. AWS IoT

Solução de problemas

- Se o Windows não reconhecer o dispositivo`VCOM`, instale o driver da porta serial do NuMaker Windows a partir do link Nu-Link USB Driver v1.6.
- Se você conectar o dispositivo ao Keil MDK (IDE) por meio do Nu-Link, verifique se o switch MSG (No.4 do ISW1 no ICE) está desativado, conforme mostrado.



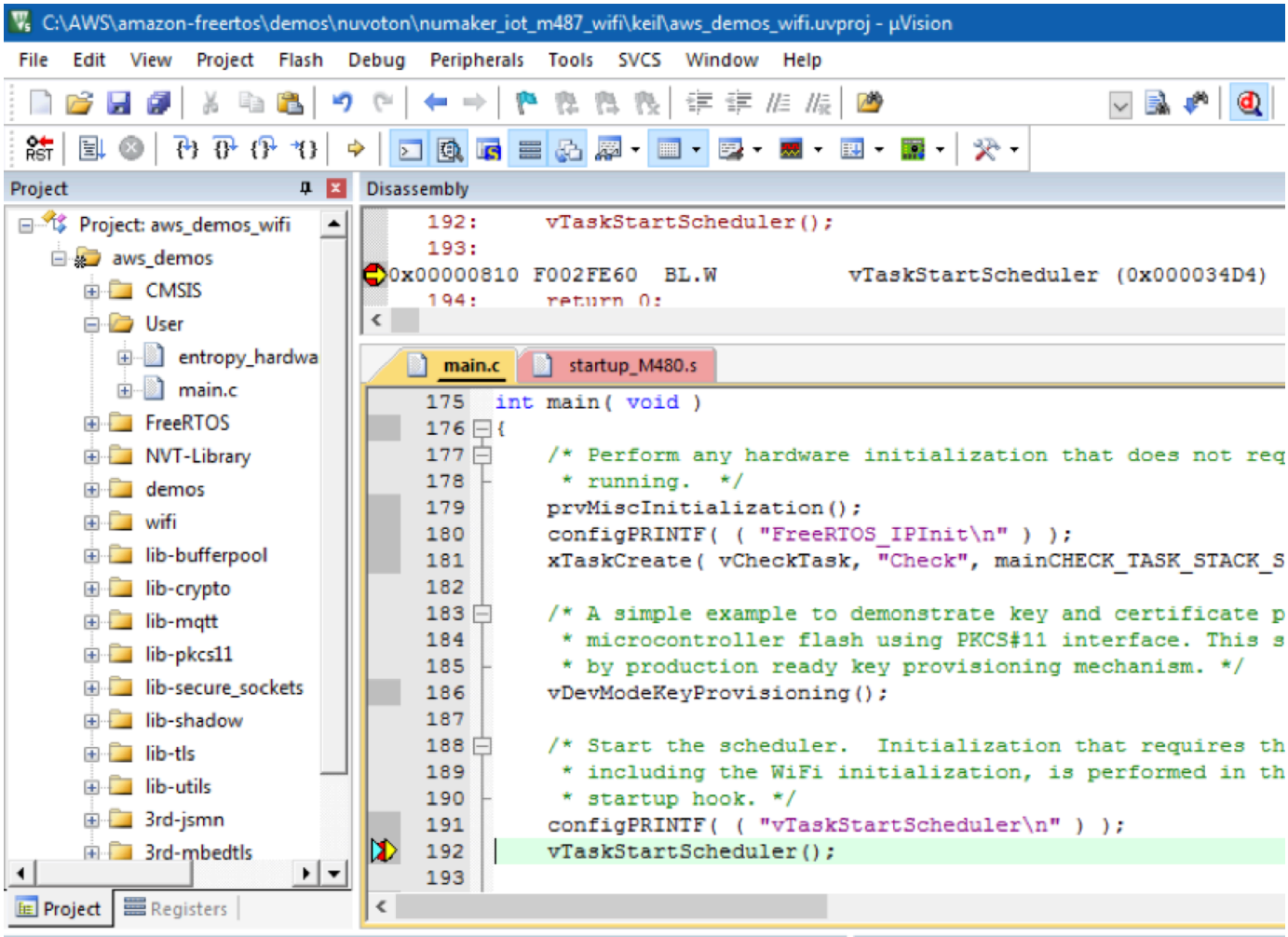
Se você tiver problemas para configurar seu ambiente de desenvolvimento ou se conectar à placa, entre em contato com a Nuvoton.

Depuração de projetos do FreeRTOS no Keil µVision

Para iniciar uma sessão de depuração no Keil µVision

1. Abra o Keil µVision.
2. Siga as etapas para compilar o projeto de demonstração do FreeRTOS em Compilação e execução do projeto de demonstração do FreeRTOS.
3. No menu Debug (Depurar), escolha Start/Stop Debug Session (Iniciar/interromper sessão de depuração).

   A janela Call Stack+Locals (Chamar pilha+locais) é exibida quando você inicia uma sessão de depuração. O µVision exibe a demonstração na placa, executa a demonstração e para no início da função `main()`.
4. Defina os pontos de interrupção no código-fonte do projeto e execute-o. O projeto deve parecer com algo semelhante ao seguinte:

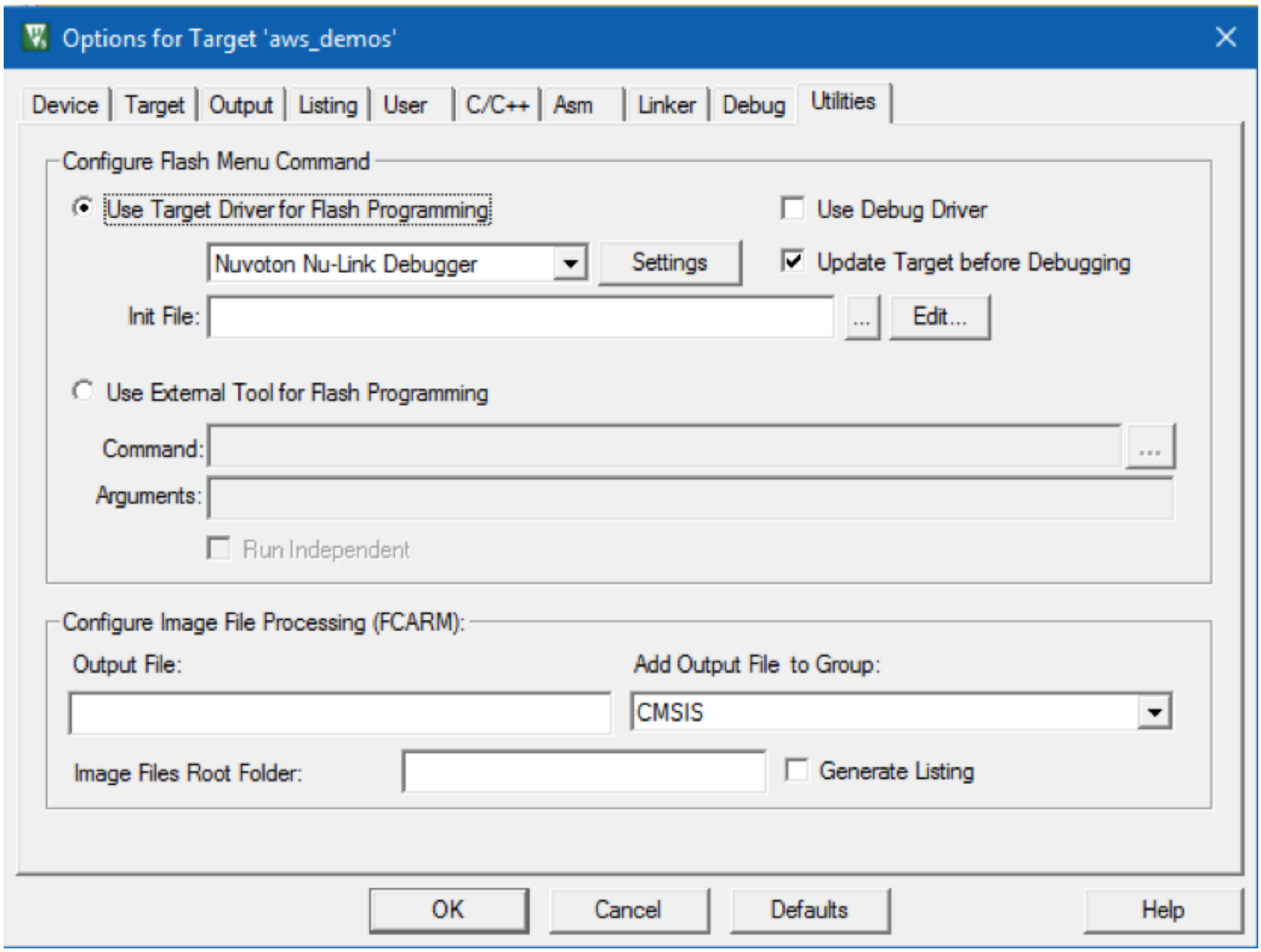


Solução de problemas de configurações de depuração do µVision

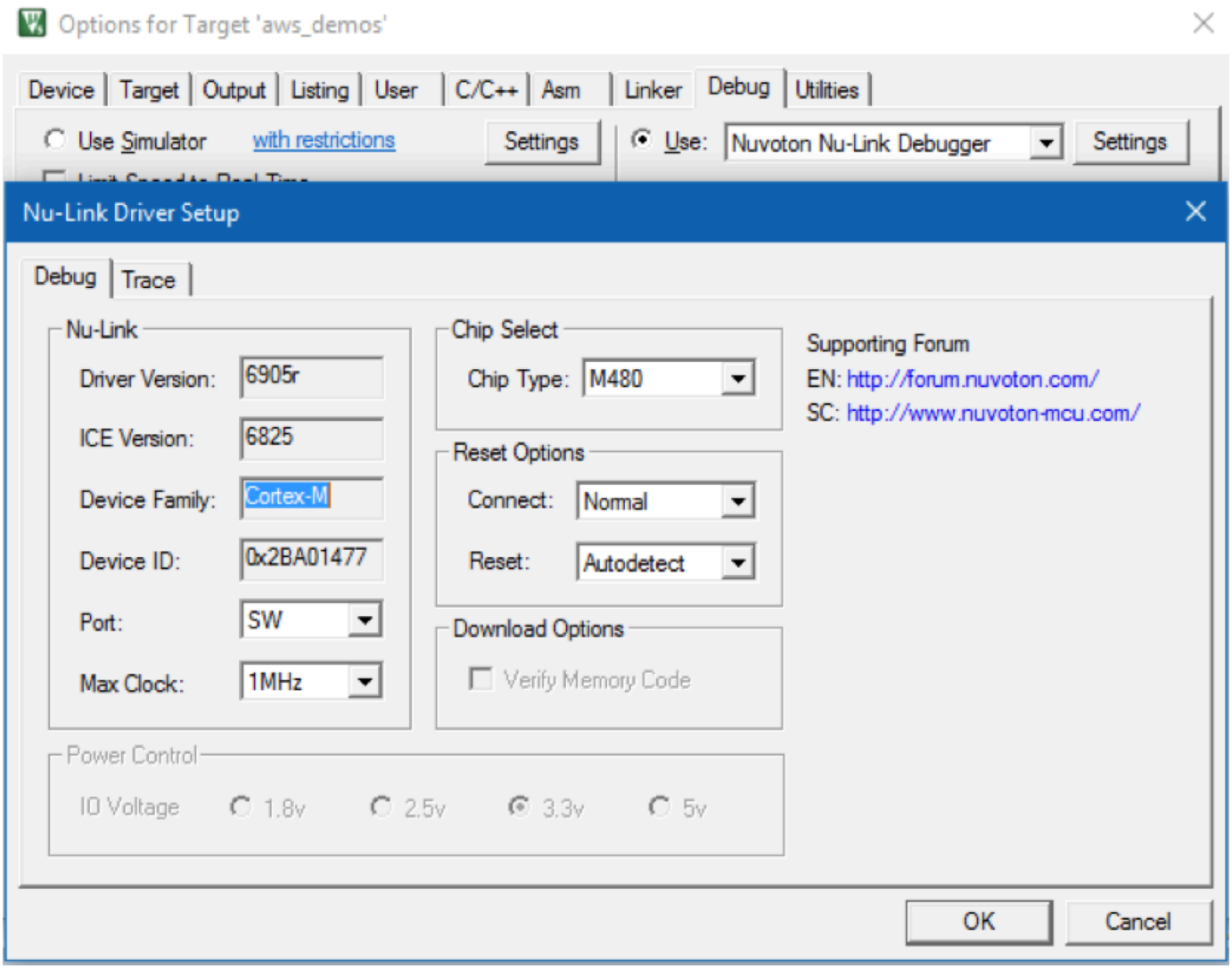
Se você encontrar problemas ao depurar uma aplicação, verifique se as configurações de depuração estão definidas corretamente no Keil µVision.

Para verificar se as configurações de depuração do µVision estão corretas

1. Abra o Keil µVision.
2. Clique com o botão direito do mouse no projeto `aws_demo` no IDE e escolha Options (Opções).
3. Na guia Utilities (Utilitários) verifique se Use Target Driver for Flash Programming (Usar o driver de destino para programação flash) está selecionado e se Nuvoton Nu-Link Debugger (Depurador Nuvoton Nu-Link) está definido como o driver de destino.



4. Na guia Debug (Depurar), ao lado de Nuvoton Nu-Link Debugger (Depurador Nuvoton Nu-Link), escolha Settings (Configurações).



5. Verifique se Chip Type (Tipo de chip) está definido como M480.

Introdução ao módulo de IoT NXP LPC54 018

> **Important**
>
> Essa integração de referência está hospedada no RTOS repositório Amazon-Free, que está obsoleto. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um RTOS projeto gratuito existente com base no repositório Amazon-FreeRTOS, agora obsoleto, consulte o. Guia de migração do repositório Github do Amazon FreeRTOS

Este tutorial fornece instruções para começar a usar o Módulo IoT NXP LPC54 018. Se você não tiver um módulo IoT NXP LPC54 018, visite AWS o Catálogo de dispositivos do parceiro para comprar um de nosso parceiro. Use um USB cabo para conectar seu módulo IoT NXP LPC54 018 ao seu computador.

Antes de começar, você deve configurar AWS IoT e RTOS fazer o download gratuito para conectar seu dispositivo à AWS nuvem. Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de RTOS download gratuito é chamado de`freertos`.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Conectar sua placa a uma máquina host.
2. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
3. Compilação cruzada de um aplicativo de RTOS demonstração gratuito em uma imagem binária.
4. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.

Configurar o NXP hardware

Para configurar o NXP LPC54 018

- Conecte seu computador à USB porta do NXP LPC54 018.

Para configurar o JTAG depurador

Você precisa de um JTAG depurador para iniciar e depurar seu código em execução na placa 018. NXP LPC54 O Free RTOS foi testado usando um OM4 módulo IoT 0006. Para obter mais informações sobre os depuradores compatíveis, consulte o Manual do usuário do Módulo NXP LPC54 IoT 018, disponível na página do produto do Módulo IoT 0007 OM4 018. LPC54

1. Se você estiver usando um depurador do módulo IoT OM4 0006, use um cabo conversor para conectar o conector de 20 pinos do depurador ao conector de 10 pinos no módulo IoT. NXP
2. Conecte o NXP LPC54 018 e o OM4 0006 IoT Module Debugger às USB portas do seu computador usando mini cabos. USB USB

Configuração do ambiente de desenvolvimento

O Free RTOS suporta dois IDEs para o módulo IoT NXP LPC54 018IAR: Embedded Workbench e. MCUXpresso

Antes de começar, instale um dessesIDEs.

Para instalar o IAR Embedded Workbench para ARM

1. Navegue até o IAREmbedded Workbench ARM e baixe o software.

   > **Note**
   >
   > IARO Embedded Workbench for ARM requer o Microsoft Windows.

2. Execute o instalador e siga as instruções.
3. No Assistente de Licença, escolha Registrar com IAR sistemas para obter uma licença de avaliação.
4. Coloque o carregador de inicialização no dispositivo antes de tentar executar qualquer demonstração.

Para instalar a MCUXpresso partir de NXP

1. Baixe e execute o MCUXpresso instalador em NXP.

   > **Note**
   >
   > As versões 10.3 e posterior são compatíveis.

2. Navegue até MCUXpressoSDKe escolha Crie seuSDK.

   > **Note**
   >
   > As versões 2.5 e posterior são compatíveis.

3. Escolha Select Development Board (Selecionar placa de desenvolvimento).
4. Em Select Development Board (Selecionar placa de desenvolvimento), em Search by Name (Pesquisar por nome), insira **`LPC54018-IoT-Module`**.
5. Em Placas, escolha LPC54018-IoT-Module.

6. Verifique os detalhes do hardware e escolha Construir MCUXepresso SDK.
7. O SDK para Windows usando o já MCUXpresso IDE está construído. Escolha BaixarSDK. Se você estiver usando outro sistema operacional, em Host OS, escolha seu sistema operacional e escolha Baixar SDK.
8. Inicie MCUXpresso IDE o e escolha a SDKs guia Instalado.
9. Arraste e solte o SDK arquivo baixado na SDKs janela Instalado.

Se você tiver problemas durante a instalação, consulte NXPSupport ou NXPDeveloper Resources.

Monitorando MQTT mensagens na nuvem

Antes de executar o projeto de RTOS demonstração gratuito, você pode configurar o MQTT cliente no AWS IoT console para monitorar as mensagens que seu dispositivo envia para a AWS nuvem.

Para assinar o MQTT tópico com o AWS IoT MQTT cliente

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Testar e, em seguida, escolha cliente MQTT de teste para abrir o MQTT cliente.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Crie e execute o projeto de RTOS demonstração gratuita

Importe a RTOS demonstração gratuita para o seu IDE

Para importar o código de RTOS amostra grátis para o IAR Embedded Workbench IDE

1. Abra o IAR Embedded Workbench e, no menu Arquivo, escolha Abrir espaço de trabalho.
2. Na caixa de texto search-directory, digite `projects/nxp/lpc54018iotmodule/iar/aws_demos` e escolha aws_demos.eww.
3. No menu Project (Projeto), escolha Rebuild All (Recriar tudo).

Para importar o código RTOS de amostra grátis para o MCUXpresso IDE

1. Abra eMCUXpresso, no menu Arquivo, escolha Abrir projetos do sistema de arquivos.
2. Na caixa de texto Directory (Diretório), digite `projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos` e escolha Finish (Concluir)
3. No menu Project (Projeto), escolha Build All (Criar tudo).

Execute o projeto de RTOS demonstração gratuito

Para executar o projeto de RTOS demonstração gratuito com o IAR Embedded Workbench IDE

1. No seuIDE, no menu Projeto, escolha Criar.
2. No menu Project (Projeto), escolha Download and Debug (Fazer download e depurar).
3. No menu Debug (Depurar), escolha Start Debugging (Iniciar depuração).
4. Quando o depurador parar no ponto de interrupção em `main`, no menu Debug (Depurar), escolha Go (Ir).

> **Note**
>
> Se uma caixa de diálogo J-Link Device Selection (Seleção do dispositivo J-Link) for aberta, escolha OK para continuar. Na caixa de diálogo Target Device Settings (Configurações do dispositivo de destino), escolha Unspecified (Não especificado), Cortex-M4 e, em seguida, OK. Você precisa fazer isso apenas uma vez.

Para executar o projeto de RTOS demonstração gratuito com o MCUxpresso IDE

1. No seuIDE, no menu Projeto, escolha Construir.
2. Se esta for a primeira depuração, escolha o projeto `aws_demos` e, na barra de ferramentas Debug (Depurar), escolha o botão de depuração azul.
3. Os testes de depuração detectados são exibidos. Selecione o teste que deseja usar e, em seguida, escolha OK para iniciar a depuração.

> **Note**
>
> Quando o depurador parar no ponto de interrupção em `main()`, pressione o botão de reiniciar depuração

uma vez para redefinir a sessão de depuração. (Isso é necessário devido a um bug com o MCUXpresso depurador do módulo 018-IoT). NXP54

4. Quando o depurador parar no ponto de interrupção em `main()`, no menu Debug (Depurar), escolha Go (Ir).

Solução de problemas

Para obter informações gerais sobre solução de problemas sobre como começar a usar o FreeRTOS, consulteSolução de problemas de conceitos básicos.

Conceitos básicos do Renesas Starter Kit+ para RX65N-2MB

Important

Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar o Renesas Starter Kit+ para RX65N-2MB. Se você não tiver o Renesas RSK+ para RX65N-2MB, visite o Catálogo de dispositivos de parceiros e adquira um de nossos AWS parceiros.

Antes de começar, você deve configurar AWS IoT e fazer o download dos FreeRTOS para conectar seu dispositivo à nuvem. AWS Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de download do FreeRTOS é chamado *`freertos`*.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Conectar sua placa a uma máquina host.
2. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.

3. Compilar uma aplicação de demonstração do FreeRTOS de forma cruzada para uma imagem binária.
4. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.

Configuração do hardware Renesas

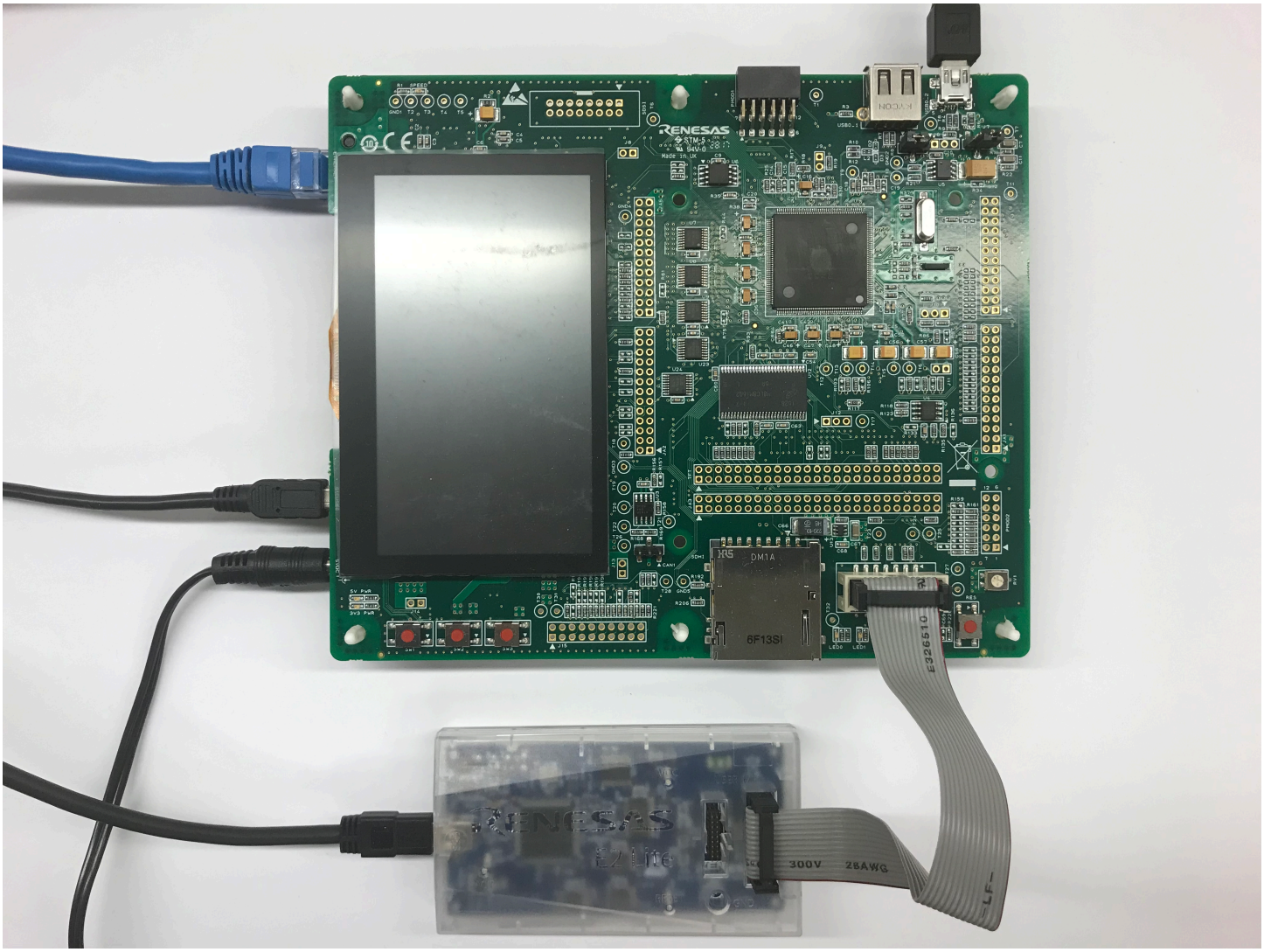
Para configurar o RSK+ para RX65N – 2 MB

1. Conecte o adaptador de energia positiva +5V ao conector PWR no RSK+para RX65N–2 MB.
2. Conecte seu computador à porta no USB2.0 FS no RSK+ para RX65N-2 MB.
3. Conecte seu computador à porta serial USB no RSK+ para RX65N-2 MB.
4. Conecte um roteador ou porta Ethernet conectada à Internet na porta Ethernet em seu RSK+ para RX65N–2 MB.

Para configurar o módulo depurador E2 Lite

1. Use o cabo chato de 14 pinos para conectar o módulo depurador E2 Lite à porta "E1/E2 Lite" no RSK+para RX65N – 2 MB.
2. Use um cabo USB para conectar o módulo depurador E2 Lite à sua máquina host. Quando o depurador E2 Lite estiver conectado à placa e ao seu computador, um LED verde "ACT" no depurador piscará.
3. Depois que o depurador estiver conectado à sua máquina host e ao RSK+para RX65N –2 MB, a instalação dos drivers do depurador E2 Lite será iniciada.

   Observe que privilégios de administrador são necessários para instalar os drivers.



Configuração do ambiente de desenvolvimento

Para definir as configurações do FreeRTOS para o RSK+ para RX65N-2 MB, use o Renesas e^{2}studio IDE e o compilador CC-RX.

> **Note**
>
> O Renesas e^{2}studio IDE e o compilador CC-RX têm suporte somente nos sistemas operacionais Windows 7, 8 e 10.

Para fazer download e instalar o e^{2}studio

1. Acesse a página de download do instalador do Renesas e^{2}studio e faça download do instalador offline.
2. Você será direcionado para uma página de login do Renesas.

   Se você tem uma conta do Renesas, insira suas credenciais de login e, em seguida, escolha Login.

   Se você não tiver uma conta, selecione Register now (Registrar-se agora), e siga as primeiras etapas de registro. Você receberá um e-mail com um link para ativar a conta do Renesas. Siga este link para concluir o registro com o Renesas e, em seguida, faça login no Renesas.
3. Depois de fazer login, faça o download do instalador do e^{2}studio em seu computador.
4. Abra o instalador e siga as etapas para a conclusão.

Para obter mais informações, consulte o e^{2}studio no site da Renesas.

Para fazer download e instalar o RX Family C/C++ Compiler Package

1. Acesse a página de download do RX Family C/C++ Compiler Package e faça download do pacote V3.00.00.
2. Abra o executável e instale o compilador.

Para obter mais informações, consulte o C/C++ Compiler Package for RX Family no site do Renesas.

> **Note**
>
> O compilador está disponível gratuitamente somente para a versão de avaliação e é válido por 60 dias. No 61º dia, você precisará obter uma chave de licença. Para obter mais informações, consulte Ferramentas de avaliação de software.

Compilação e execução de exemplos do FreeRTOS

Agora que o projeto de demonstração está configurado, você está pronto para compilar e executar o projeto em sua placa.

Compilar a demonstração do FreeRTOS no e^{2}studio

Para importar e criar a demonstração no e^{2}studio

1. Inicie o e^{2}studio no menu Iniciar.
2. Na janela Select a directory as a workspace (Selecionar um diretório como um espaço de trabalho), navegue até a pasta na qual deseja trabalhar em e selecione Launch (Iniciar).
3. A primeira vez que você abrir o e^{2}studio, a janela Toolchain Registry (Registro da cadeia de ferramentas) será aberta. Selecione Renesas Toolchains (Conjuntos de ferramentas do Renesas) e confirme que **`CC-RX v3.00.00`** está selecionado. Selecione Register (Registrar) e, em seguida, selecione OK.
4. Se você estiver abrindo o e^{2}studio pela primeira vez, a janela Code Generator Registration (Registro do gerador de código) será exibida. Escolha OK.
5. A janela Code Generator COM component register (registro do componente Gerador de código COM) será exibida. Em Por favor, reinicie o e^{2}studio para usar o Gerador de códigos, escolha OK.
6. A janela Reiniciar o e^{2}studio é exibida. Escolha OK.
7. O e^{2}studio é reiniciado. Na janela Select a directory as a workspace (Selecionar um diretório como um espaço de trabalho), selecione Launch (Iniciar).
8. Na tela de boas-vindas do e^2 studio, escolha o ícone de seta Ir para o e^2 studio do Workbench.
9. Clique com o botão direito do mouse na janela Project Explorer (Explorador de projetos) e selecione Import (Importar).
10. No assistente de importação , selecione General (Geral), Existing Projects into Workspace (Projetos existentes no espaço de trabalho) e, em seguida, selecione Next (Próximo).
11. Selecione Browse (Procurar), localize o diretório `projects/renesas/rx65n-rsk/e2studio/aws_demos` e selecione Finish (Concluir).
12. No menu Project (Projeto), selecione Project (Projeto), Build All (Criar tudo).

    O console de compilação emite uma mensagem de aviso de que o Gerenciador de licenças não está instalado. Você pode ignorar essa mensagem, a menos que tenha uma chave de licença para o CC-RX compiler. Para instalar o Gerenciador de licenças, consulte a página de download do Gerenciador de licenças.

Monitoramento de mensagens MQTT na nuvem

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console para monitorar AWS IoT as mensagens que seu dispositivo envia para a nuvem. AWS

Para assinar o tópico MQTT com o cliente AWS IoT MQTT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Execução do projeto FreeRTOS

Para executar o projeto no e^{2}studio

1. Confirme que você conectou o módulo depurador E2 Lite ao seu RSK+para RX65N – 2 MB
2. No menu superior, selecione Run (Executar), Debug Configurations (Configurações de depuração).
3. Expanda o Renesas GDB Hardware Debugging e escolha aws_demos. HardwareDebug
4. Selecione a guia Debugger (Depurador) e, em seguida, selecione a guia Connection Settings (Configurações de conexão) . Confirme se as configurações de conexão estão corretas.
5. Selecione Debug (Depurar) para fazer download do código para sua placa e começar a depuração.

   Pode ser solicitado `e2-server-gdb.exe` por um aviso de firewall. Verifique as Private networks, such as my home or work network (Redes privadas, como minha casa ou rede de trabalho) e, em seguida, selecione Allow access (Permitir acesso).
6. O e^{2}studio pode solicitar a alteração para a Renesas Debug Perspective (Perspectiva de depuração do Renesas). Escolha Sim.

   O LED verde "ACT" no depurador E2 Lite ficará aceso.

7. Depois que o download do código for feito para a placa, selecione Resume (Retomar) para executar o código até a primeira linha da função principal. Selecione Resume (Retomar) novamente para executar o restante do código.

Para os projetos mais recentes lançados pela Renesas, consulte a `renesas-rx` bifurcação do `amazon-freertos` repositório em. GitHub

Solução de problemas

Para obter mais informações sobre soluções de problemas gerais sobre os Conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Introdução ao STMicroelectronics STM32L4 Discovery Kit IoT Node

> **Important**
>
> Essa integração de referência está hospedada no RTOS repositório Amazon-Free, que está obsoleto. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um RTOS projeto gratuito existente com base no repositório Amazon-FreeRTOS, agora obsoleto, consulte o. Guia de migração do repositório Github do Amazon FreeRTOS

Este tutorial fornece instruções para começar a usar o STMicroelectronics STM32L4 Discovery Kit IoT Node. Se você ainda não tem o STMicroelectronics STM32L4 Discovery Kit IoT Node, visite o Catálogo de dispositivos do AWS parceiro para comprar um de nosso parceiro.

Certifique-se de que você tenha instalado o firmware Wi-Fi mais recente. Para baixar o firmware Wi-Fi mais recente, consulte Node IoT do kit STM32L4 Discovery, conexão sem fio de baixa potência, Bluetooth Low Energy, S NFCubGHz, Wi-Fi. Em Recursos binários, escolha Atualização do firmware do módulo Wi-Fi Inventek ISM 43362 (leia o arquivo readme para obter instruções).

Antes de começar, você deve configurar AWS IoT seu RTOS download gratuito e o Wi-Fi para conectar seu dispositivo à AWS nuvem. Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de RTOS download gratuito é chamado de*`freertos`*.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilação cruzada de um aplicativo de RTOS demonstração gratuito em uma imagem binária.
3. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.

Configuração do ambiente de desenvolvimento

Instale o System Workbench para STM32

1. Navegue até STM32Open.org.
2. Registre-se na STM32 página do Open. Você precisa fazer login para fazer download do System Workbench.
3. Navegue até o System Workbench para obter o STM32 instalador para baixar e instalar o System Workbench.

Se você tiver problemas durante a instalação, consulte o FAQs site do System Workbench.

Crie e execute o projeto de RTOS demonstração gratuito

Importe a RTOS demonstração gratuita para o STM32 System Workbench

1. Abra o STM32 System Workbench e insira um nome para um novo espaço de trabalho.
2. No menu File (Arquivo), escolha Import (Importar). Expanda General (Geral), escolha Existing Projects into Workspace (Projetos existentes no espaço de trabalho) e, em seguida, Next (Próximo).
3. Em Select Root Directory (Selecionar diretório raiz), digite `projects/st/stm32l475_discovery/ac6/aws_demos`.
4. O projeto `aws_demos` deve ser selecionado por padrão.
5. Escolha Concluir para importar o projeto para o STM32 System Workbench.
6. No menu Project (Projeto), escolha Build All (Criar tudo). Confirme se o projeto foi compilado sem erros.

Monitorando MQTT mensagens na nuvem

Antes de executar o projeto de RTOS demonstração gratuito, você pode configurar o MQTT cliente no AWS IoT console para monitorar as mensagens que seu dispositivo envia para a AWS nuvem.

Para assinar o MQTT tópico com o AWS IoT MQTT cliente

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Testar e, em seguida, escolha cliente MQTT de teste para abrir o MQTT cliente.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Execute o projeto de RTOS demonstração gratuito

1. Use um USB cabo para conectar o Node IoT do STMicroelectronics STM32L4 Discovery Kit ao seu computador. (Verifique a documentação do fabricante que veio com sua placa para saber a USB porta correta a ser usada.)
2. No Project Explorer, clique com o botão direito do mouse`aws_demos`, escolha Depurar como e escolha Aplicativo Ac6 STM32 C/C++.

   Se um erro de depuração ocorrer na primeira vez que uma sessão de depuração for iniciada, siga estas etapas:

   1. No STM32 System Workbench, no menu Executar, escolha Configurações de depuração.
   2. Escolha aws_demos Debug (Depuração de aws_demos). (Talvez seja necessário expandir a STM32depuração Ac6.)
   3. Escolha a guia Debugger (Depurador).
   4. Em Configuration Script (Script de configurações), escolha Show Generator Options (Mostrar opções do gerador).
   5. Em Mode Setup (Configuração de modo), defina Reset Mode (Modo de redefinição) como Software System Reset (Redefinição do sistema de software). Escolha Apply e selecione Debug.
3. Quando o depurador parar no ponto de interrupção em `main()`, no menu Run (Executar), escolha Resume (Continuar).

Usando CMake com o Free RTOS

Se você preferir não usar um IDE RTOS desenvolvimento gratuito, você pode usar como alternativa CMake para criar e executar os aplicativos de demonstração ou aplicativos que você desenvolveu usando editores de código e ferramentas de depuração de terceiros.

Primeiro, crie uma pasta para conter os arquivos de compilação gerados (*build-folder*).

Use o comando a seguir para gerar os arquivos de compilação:

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-folder
```

Se não `arm-none-eabi-gcc` estiver no caminho do shell, você também precisará definir a `AFR_TOOLCHAIN_PATH` CMake variável. Por exemplo:

```
-D AFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

Para obter mais informações sobre como usar CMake com o FreeRTOS, consulteUso da CMake com o FreeRTOS.

Solução de problemas

Se você ver o seguinte na UART saída do aplicativo de demonstração, precisará atualizar o firmware do módulo Wi-Fi:

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

Para baixar o firmware Wi-Fi mais recente, consulte Node IoT do kit STM32L4 Discovery, conexão sem fio de baixa potência, Bluetooth Low Energy, S NFCubGHz, Wi-Fi. Em Recursos binários, escolha o link de download para a atualização do firmware do módulo Wi-Fi Inventek ISM 43362.

Para obter informações gerais sobre solução de problemas sobre como começar a usar o FreeRTOS, consulteSolução de problemas de conceitos básicos.

Começando com a Texas Instruments CC322 0SF- LAUNCHXL

> **Important**
>
> Essa integração de referência está hospedada no RTOS repositório Amazon-Free, que está obsoleto. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um RTOS projeto gratuito existente com base no repositório Amazon-FreeRTOS, agora obsoleto, consulte o. Guia de migração do repositório Github do Amazon FreeRTOS

Este tutorial fornece instruções para começar a usar o Texas Instruments CC322 0SF-. LAUNCHXL Se você não tiver o kit de LAUNCHXL desenvolvimento CC322 0SF da Texas Instruments (TI), visite o Catálogo de dispositivos do AWS parceiro para comprar um de nosso parceiro.

Antes de começar, você deve configurar AWS IoT e RTOS fazer o download gratuito para conectar seu dispositivo à AWS nuvem. Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de RTOS download gratuito é chamado de`freertos`.

Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilação cruzada de um aplicativo de RTOS demonstração gratuito em uma imagem binária.
3. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.

Configuração do ambiente de desenvolvimento

Siga as etapas abaixo para configurar seu ambiente de desenvolvimento e começar a usar o FreeRTOS.

Observe que o Free RTOS suporta dois IDEs para o kit de LAUNCHXL desenvolvimento TI CC322 0SF: Code Composer Studio e IAR Embedded Workbench versão 8.32. Você pode usar qualquer um deles IDE para começar.

Instalar o Code Composer Studio

1. Vá até TI Code Composer Studio.

2. Faça download do instalador off-line da plataforma da máquina host (Windows, macOS ou Linux 64 bits).
3. Descompacte e execute o instalador off-line. Siga os prompts.
4. Para famílias de produtos a serem instaladas, escolha SimpleLink Wi-Fi CC32xx Wireless MCUs.
5. Na próxima página, aceite as configurações padrão para testes de depuração e, em seguida, escolha Finish (Concluir).

Se você tiver problemas ao instalar o Code Composer Studio, consulte TI Development Tools Support, Code Composer Studio FAQs e Solução de problemas. CCS

Instale o IAR Embedded Workbench

1. Baixe e execute o instalador do Windows para a versão 8.32 do IAR Embedded Workbench for. ARM Em Debug probe drivers, certifique-se de que TI XDS esteja selecionado.
2. Conclua a instalação e inicie o programa. Na página License Wizard, escolha Registrar com IAR sistemas para obter uma licença de avaliação ou use sua própria IAR licença.

Instale o SimpleLink CC322 0 SDK

Instale o SimpleLink CC3220 SDK. O SimpleLink Wi-Fi CC322 0 SDK contém drivers para o CC322 0SF programávelMCU, mais de 40 aplicativos de amostra e a documentação necessária para usar as amostras.

Instalar o Uniflash

Instale o Uniflash. CCSO Uniflash é uma ferramenta autônoma usada para programar a memória flash no chip na TI. MCUs O Uniflash tem uma linha GUI de comando e uma interface de script.

Instalação do Service Pack mais recente

1. Em seu TI CC322 0SF-LAUNCHXL, coloque o SOP jumper no conjunto central de pinos (posição = 1) e reinicie a placa.
2. Inicie o Uniflash. Se sua LaunchPad placa CC322 0SF aparecer em Dispositivos detectados, escolha Iniciar. Se sua placa não for detectada, escolha CC3220SF- na lista LAUNCHXL de placas em Nova configuração e, em seguida, escolha Iniciar Criador de Imagem.
3. Escolha New Project (Novo projeto).

4. Na página Start new project (Iniciar novo projeto), insira um nome para o projeto. Em Tipo de dispositivo, escolha CC3220SF. Em Device Mode (Modo de dispositivo), escolha Develop (Desenvolver) e, em seguida, Create Project (Criar projeto).
5. No lado direito da janela da aplicação Uniflash, escolha Connect (Conectar).
6. Na coluna à esquerda, selecione Advanced (Avançado), Files (Arquivos) e, em seguida, Service Pack (Pacotes de serviços).
7. Escolha Procurar e, em seguida, navegue até onde você instalou o CC322 0SF SimpleLink SDK. O service pack está localizado em `ti/simplelink_cc32xx_sdk_`*`VERSION`*`/tools/cc32xx_tools/servicepack-cc3x20/sp_`*`VERSION`*`.bin`.
8. Selecione o botão Burn (Gravar)

   ()

   e depois selecione Program Image (Create & Program) (Imagem do programa (criar e programar)) para instalar o pacote de serviços. Lembre-se de colocar o SOP jumper de volta na posição 0 e reinicializar a placa.

Configuração de provisionamento de Wi-Fi

Para definir as configurações de Wi-Fi para sua placa, execute uma das seguintes ações:

- Configure o aplicativo de RTOS demonstração gratuito descrito emConfiguração das demonstrações do FreeRTOS.
- Use SmartConfigda Texas Instruments.

Crie e execute o projeto de RTOS demonstração gratuito

Crie e execute o projeto de RTOS demonstração gratuito no TI Code Composer

Para importar a RTOS demonstração gratuita para o TI Code Composer

1. Abra o TI Code Composer e escolha OK para aceitar o nome padrão do espaço de trabalho.
2. Na página Getting Started (Conceitos básicos), selecione Import Project (Importar projeto).
3. Em Select search-directory (Selecionar diretório de pesquisa), digite `projects/ti/cc3220_launchpad/ccs/aws_demos`. O projeto `aws_demos` deve ser selecionado por padrão. Para importar o projeto para o TI Code Composer, escolha Finish (Concluir).
4. No Project Explorer, clique duas vezes em aws_demos para tornar o projeto ativo.

5. Em Project (Projeto), escolha Build Project (Criar projeto) para garantir que o projeto seja compilado sem erros ou avisos.

Para executar a RTOS demonstração gratuita no TI Code Composer

1. Certifique-se de que o jumper Sense On Power (SOP) em seu Texas Instruments CC322 0SF-LAUNCHXL esteja na posição 0. Para obter mais informações, consulte SimpleLink Wi-Fi CC3x20, Guia do usuário do processador de CC3x3x rede.
2. Use um USB cabo para conectar sua Texas Instruments CC322 0SF- LAUNCHXL ao seu computador.
3. No explorador de projeto, certifique-se de que o `CC3220SF.ccxml` esteja selecionado como a configuração de destino ativa. Para ativá-la, clique com o botão direito do mouse no arquivo e selecione Set as active target configuration (Definir como configuração de destino ativa).
4. No TI Code Composer, em Run (Executar), selecione Debug (Depurar).
5. Quando o depurador parar no ponto de interrupção em `main()`, vá para o menu Run (Executar) e escolha Resume (Continuar).

Monitorando MQTT mensagens na nuvem

Antes de executar o projeto de RTOS demonstração gratuito, você pode configurar o MQTT cliente no AWS IoT console para monitorar as mensagens que seu dispositivo envia para a AWS nuvem.

Para assinar o MQTT tópico com o AWS IoT MQTT cliente

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Testar e, em seguida, escolha cliente MQTT de teste para abrir o MQTT cliente.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Crie e execute um projeto de RTOS demonstração gratuito no IAR Embedded Workbench

Para importar a RTOS demonstração gratuita para o IAR Embedded Workbench

1. Abra o IAR Embedded Workbench, escolha Arquivo e, em seguida, escolha Open Workspace.
2. Navegue até `projects/ti/cc3220_launchpad/iar/aws_demos`, escolha aws_demos.eww e escolha OK.
3. Clique com o botão direito do mouse no nome do projeto (`aws_demos`) e, em seguida, selecione Make (Fazer).

Para executar a RTOS demonstração gratuita no IAR Embedded Workbench

1. Certifique-se de que o jumper Sense On Power (SOP) em seu Texas Instruments CC322 0SF-LAUNCHXL esteja na posição 0. Para obter mais informações, consulte SimpleLink Wi-Fi CC3x20, Guia do usuário do processador de CC3x3x rede.
2. Use um USB cabo para conectar sua Texas Instruments CC322 0SF- LAUNCHXL ao seu computador.
3. Compile o projeto novamente.

   Para recriar o projeto, no menu Project (Projeto), escolha Make (Fazer).
4. No menu Project (Projeto), escolha Download and Debug (Fazer download e depurar). Você pode ignorar “Aviso: falha na inicialização” EnergyTrace, se for exibido. Para obter mais informações sobre EnergyTrace, consulte MSP EnergyTrace Tecnologia.
5. Quando o depurador parar no ponto de interrupção em `main()`, vá para o menu Debug (Depurar) e escolha Go (Ir).

Usando CMake com o Free RTOS

Se você preferir não usar um IDE RTOS desenvolvimento gratuito, você pode usar como alternativa CMake para criar e executar os aplicativos de demonstração ou aplicativos que você desenvolveu usando editores de código e ferramentas de depuração de terceiros.

Para criar a RTOS demonstração gratuita com CMake

1. Crie uma pasta para conter os arquivos de compilação gerados (*`build-folder`*).

2. Certifique-se de que seu caminho de pesquisa (variável de PATH ambiente $) contenha a pasta em que o binário do CGT compilador de TI está localizado (por exemploC:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin).

   Se você estiver usando o ARM compilador TI com sua placa TI, use o seguinte comando para gerar arquivos de compilação a partir do código-fonte:

   ```
   cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S freertos -B build-folder
   ```

Para obter mais informações, consulte Uso da CMake com o FreeRTOS.

Solução de problemas

Se você não vê mensagens no MQTT cliente do AWS IoT console, talvez seja necessário definir as configurações de depuração da placa.

Para configurar as configurações de depuração para placas de TI

1. No Code Composer, no Project Explorer, escolha aws_demos.
2. No menu Run (Executar), escolha Debug Configurations (Configurações de depuração).
3. No painel de navegação, selecione aws_demos.
4. Na guia Target (Destino), em Connection Options (Opções de conexão), escolha Reset the target on a connect (Redefinir o destino em uma conexão).
5. Escolha Aplicar e selecione Fechar.

Se essas etapas não funcionarem, verifique a saída do programa no terminal serial. Você deve ver um texto que indica a origem do problema.

Para obter informações gerais sobre solução de problemas sobre como começar a usar o FreeRTOS, consulteSolução de problemas de conceitos básicos.

Conceitos básicos do simulador de dispositivo do Windows

Este tutorial fornece instruções para começar a usar o simulador de dispositivos do Windows com o FreeRTOS.

Antes de começar, você deve configurar o AWS IoT e seu download do FreeRTOS para conectar seu dispositivo à nuvem da AWS. Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de download do FreeRTOS é chamado *freertos*.

O FreeRTOS é liberado como um arquivo zip que contém as bibliotecas e aplicativos de exemplo para a plataforma especificada. Para executar os exemplos em uma máquina Windows, faça download das bibliotecas e dos exemplos migrados para executar no Windows. Esse conjunto de arquivos é conhecido como simulador do FreeRTOS para Windows.

> **Note**
>
> Este tutorial não pode ser executado com êxito em instâncias Windows do Amazon EC2.

Configuração do ambiente de desenvolvimento

1. Instale a versão mais recente do Npcap. Selecione o "modo compatível com a API WinPcap" durante a instalação.
2. Instale o Microsoft Visual Studio.

   As versões 2017 e 2019 do Visual Studio funcionam. Todas as edições dessas versões do Visual Studio são compatíveis (Community, Professional ou Enterprise).

   Além do IDE, instale o componente Desktop development with C++ (Desenvolvimento de desktop com C++).

   Instale a versão mais recente do SDK do Windows 10. Você pode escolher isso na seção Optional (Opcional) do componente Desktop development with C++ (Desenvolvimento de desktop com C++).
3. Verifique se você tem uma conexão Ethernet fixa ativa.
4. (Opcional) Se você quiser usar o sistema de compilação com base em CMake para criar seus projetos do FreeRTOS, instale a versão mais recente da CMake. O FreeRTOS exige a versão 3.13 ou posterior da CMake.

Monitoramento de mensagens MQTT na nuvem

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console do AWS IoT para monitorar as mensagens enviadas pelo dispositivo para a nuvem da AWS.

Para assinar o tópico MQTT com o cliente MQTT do AWS IoT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Quando o projeto de demonstração for executado com êxito em seu dispositivo, você verá "Olá, mundo!". enviado várias vezes para o tópico em que você assinou.

Compilação e execução do projeto de demonstração do FreeRTOS

Você pode usar o Visual Studio ou CMake para compilar projetos do FreeRTOS.

Compilação e execução do projeto de demonstração do FreeRTOS com o IDE do Visual Studio

1. Carregue o projeto no Visual Studio.

   No Visual Studio, no menu File (Arquivo), escolha Open (Abrir). Escolha File/Solution (Arquivo/solução), navegue até o arquivo `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` e escolha Open (Abrir).
2. Defina um novo destino para o projeto de demonstração.

   O projeto de demonstração fornecido depende do SDK do Windows, mas ele não tem uma versão do SDK do Windows especificada. Por padrão, o IDE pode tentar compilar a demonstração com uma versão do SDK que não está presente em sua máquina. Para definir a versão do SDK do Windows, clique com o botão direito do mouse em `aws_demos` e escolha Retarget Projects (Definir novos destinos para os projetos). Isso abre a janela Review Solution Actions (Revisar ações de solução). Escolha uma versão do SDK do Windows que esteja presente em sua máquina (o valor inicial na lista suspensa está bom) e escolha OK.
3. Crie e execute o projeto.

No menu Build (Criar), escolha Build Solution (Criar solução) e certifique-se de que a solução seja criada sem erros ou avisos. Escolha Debug (Depurar), Start Debugging (Começar a depurar) para executar o projeto. Na primeira execução, será necessário selecionar uma interface de rede.

Criação e execução do projeto de demonstração do FreeRTOS com CMake

Recomendamos que você use a GUI do CMake em vez da ferramenta de linha de comando do CMake para criar o projeto de demonstração do simulador do Windows.

Depois de instalar o CMake, abra a GUI do CMake. No Windows, ele pode ser encontrado no menu Iniciar, em CMake, CMake (cmake-gui).

1. Defina o diretório de código-fonte do FreeRTOS.

   Na GUI, defina o diretório de código-fonte do FreeRTOS (*`freertos`*) como No qual está o código-fonte.

   Defina *`freertos`*`/build` como Where to build the binaries (Onde compilar os binários).

2. Configure o projeto do CMake.

   Na GUI do CMake, escolha Add Entry (Adicionar entrada), na janela Add Cache Entry (Adicionar entrada de cache), defina os seguintes valores:

   Nome

   AFR_BOARD

   Tipo

   STRING

   Valor

   pc.windows

   Descrição

   (Optional)

3. Selecione Configurar. Se o CMake solicitar que você crie o diretório de compilação, escolha Yes (Sim) e selecione um gerador em Specify the generator for this project (Especificar o gerador

para esse projeto). Recomendamos usar o Visual Studio como gerador, mas o Ninja também é compatível. (Observe que, ao usar o Visual Studio 2019, a plataforma deve ser definida como Win32, em vez de sua configuração padrão.) Mantenha as outras opções de gerador inalteradas e escolha Concluir.

4. Gere e abra o projeto CMake.

   Após configurar o projeto, a GUI do CMake mostra todas as opções disponíveis para o projeto gerado. Para os fins deste tutorial, você pode deixar as opções em seus valores padrão.

   Escolha Generate (Gerar) para criar uma solução do Visual Studio e escolha Open Project (Abrir projeto) para abrir o projeto no Visual Studio.

   No Visual Studio, clique com o botão direito do mouse no projeto `aws_demos` e escolha Set as StartUp Project (Definir como projeto de inicialização). Isso permite que você crie e execute o projeto. Na primeira execução, será necessário selecionar uma interface de rede.

Para obter mais informações sobre o uso do CMake com o FreeRTOS, consulte Uso da CMake com o FreeRTOS.

Configuração da interface de rede

Na primeira execução do projeto de demonstração, você deve selecionar a interface de rede a ser usada. O programa conta suas interfaces de rede. Encontre o número da interface Ethernet fixa. O resultado deve ser semelhante ao seguinte:

```
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.
```

Depois de identificar o número da interface Ethernet fixa, feche a janela do aplicativo. No exemplo anterior, o número a ser usado é 1.

Abra `FreeRTOSConfig.h` e defina `configNETWORK_INTERFACE_TO_USE` como o número que corresponde à sua interface de rede fixa.

> **Important**
>
> Somente interfaces Ethernet são compatíveis. Não há suporte para Wi-Fi.

Solução de problemas

Solução de problemas comuns no Windows

Você poderá encontrar o seguinte erro ao tentar criar o projeto de demonstração com o Visual Studio:

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual
 Studio solution.
```

O projeto deve ser direcionado a uma versão do SDK do Windows presente no seu computador.

Para obter mais informações sobre soluções de problemas gerais sobre os conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Introdução ao kit de IoT industrial Xilinx Avnet MicroZed

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial fornece instruções para começar a usar o kit de IoT MicroZed industrial Xilinx Avnet. Se você não tiver o kit de IoT MicroZed industrial Xilinx Avnet, visite o Catálogo de dispositivos AWS do parceiro para comprar um de nosso parceiro.

Antes de começar, você deve configurar AWS IoT e fazer o download dos FreeRTOS para conectar seu dispositivo à nuvem. AWS Para obter instruções, consulte Primeiras etapas. Neste tutorial, o caminho para o diretório de download do FreeRTOS é chamado *`freertos`*.

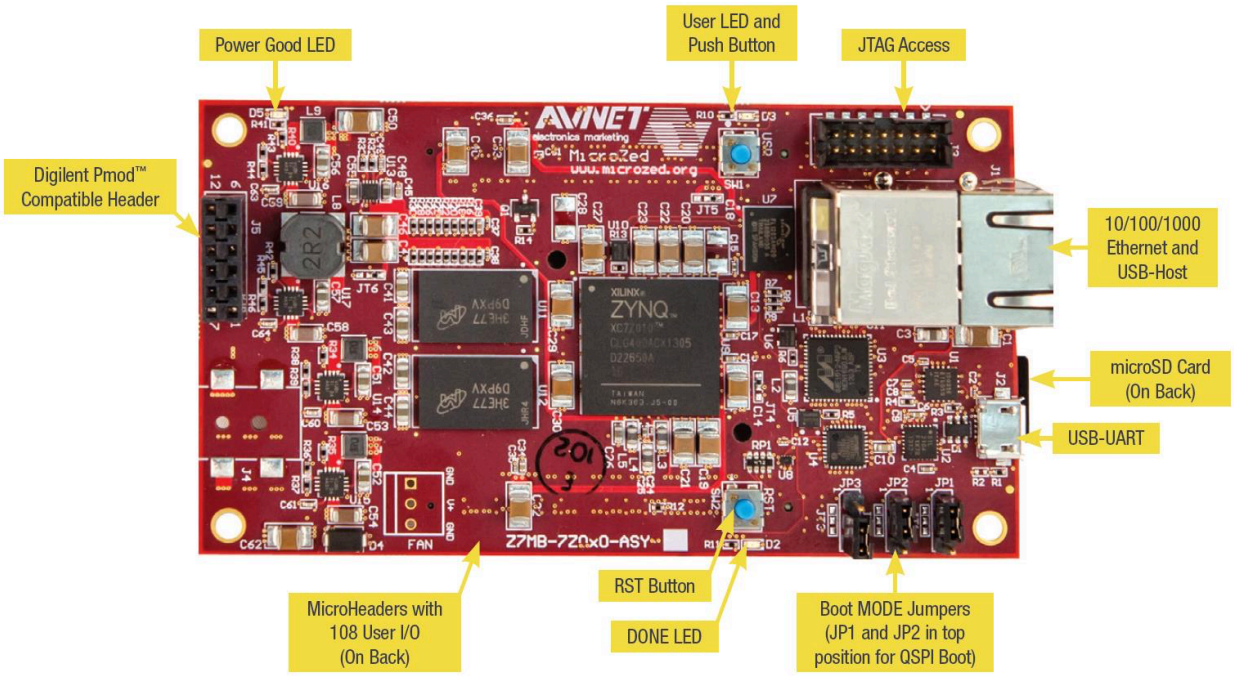
Visão geral

Este tutorial contém instruções para as seguintes etapas iniciais:

1. Conectar sua placa a uma máquina host.
2. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
3. Compilar uma aplicação de demonstração do FreeRTOS de forma cruzada para uma imagem binária.
4. Carregar a imagem binária do aplicativo em sua placa e executar o aplicativo.

Configurar o MicroZed hardware

O diagrama a seguir pode ser útil ao configurar o MicroZed hardware:



Para configurar o MicroZed quadro

1. Conecte seu computador à porta USB-UART da sua placa. MicroZed
2. Conecte seu computador à porta de acesso JTAG em sua MicroZed placa.

3. Conecte um roteador ou porta Ethernet conectada à Internet à porta Ethernet e USB host da sua placa. MicroZed

Configuração do ambiente de desenvolvimento

Para definir as configurações do FreeRTOS para o kit, você deve usar MicroZed o Xilinx Software Development Kit (XSDK). O XSDK é compatível com Windows e Linux.

Download e instalação do XSDK

Para instalar o software Xilinx, é necessário uma conta do Xilinx gratuita.

Para fazer download do XSDK

1. Acesse a página de download do Software Development Kit Standalone WebInstall Client.
2. Escolha a opção adequada para o seu sistema operacional.
3. Você será direcionado para uma página de login do Xilinx.

   Se você tem uma conta do Xilinx, insira suas credenciais de login e, em seguida, faça login.

   Se você não tiver uma conta, escolha Create your account. Depois de se registrar, você receberá um e-mail com um link para ativar a conta do Xilinx.
4. Na página Name and Address Verification, insira suas informações e, em seguida, selecione Next. O download deverá estar pronto para começar.
5. Salve o arquivo `Xilinx_SDK_version_os`.

Para instalar o XSDK

1. Abra o arquivo `Xilinx_SDK_version_os`.
2. Em Select Edition to Install, escolha Xilinx Software Development Kit (XSDK) e, em seguida, selecione Next.
3. Na página seguinte do assistente de instalação, em Installation Options, selecione Install Cable Drivers e, em seguida, escolha Next.

Se o computador não detectar a conexão USB-UART MicroZed do CP210x, instale manualmente os drivers VCP do CP210x USB-to-UART Bridge. Para obter instruções, consulte o Guia de instalação do CP210x USB-to-UART da Silicon Labs.

Para obter mais informações sobre o XSDK, consulte Conceitos básicos do Xilink SDK no site do Xilink.

Monitoramento de mensagens MQTT na nuvem

Antes de executar o projeto de demonstração do FreeRTOS, você pode configurar o cliente MQTT no console para monitorar AWS IoT as mensagens que seu dispositivo envia para a nuvem. AWS

Para assinar o tópico MQTT com o cliente AWS IoT MQTT

1. Faça login no console do AWS IoT.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Compilação e execução do projeto de demonstração do FreeRTOS

Abrir a demonstração do FreeRTOS no IDE do XSDK

1. Inicie o IDE do XSDK com o diretório do workspace definido como *freertos*`/projects/xilinx/microzed/xsdk`.
2. Feche a página de boas-vindas. No menu, escolha Project (Projeto) e, em seguida, desmarque Build Automatically (Compilar automaticamente).
3. No menu, escolha File (Arquivo) e, em seguida, selecione Import (Importar).
4. Na página Select (Selecionar), expanda General (Geral), escolha Existing Projects into Workspace (Projetos existentes no workspace) e, em seguida, selecione Next (Próximo).
5. Na página Importar projetos, escolha Selecionar diretório raiz e insira o diretório raiz do projeto de demonstração: *freertos*`/projects/xilinx/microzed/xsdk/aws_demos`. Para navegar até o diretório, escolha Browse (Navegar).

   Depois de especificar um diretório raiz, os projetos desse diretório aparecerão na página Import Projects (Importar projetos). Todos os projetos disponíveis são selecionados por padrão.

> **Note**
>
> Se houver um aviso no topo da página Import Projects (Importar projetos) ("Some projects cannot be imported because they already exist in the workspace." (Alguns projetos não podem ser importados pois já existem no workspace.)) ignore-o.

6. Com todos os projetos selecionados, escolha Finish (Concluir).
7. Se você não vir os projetos `aws_bsp`, `fsbl` e `MicroZed_hw_platform_0` no painel de projetos, repita as etapas anteriores a partir de #3, mas com o diretório raiz definido como *`freertos`*`/vendors/xilinx`, e importe `aws_bsp`, `fsbl` e `MicroZed_hw_platform_0`.
8. No menu, escolha Window (Janela) e, em seguida, selecione Preferences (Preferências).
9. No painel de navegação, expanda Run/Debug (Executar/depurar), escolha String Substitution (Substituição de strings) e, em seguida, selecione New (Novo).
10. Em New String Substitution Variable (Nova variável de substituição de strings), para Name (Nome), insira **`AFR_ROOT`**. Em Valor, insira o caminho raiz do *`freertos`*`/projects/xilinx/microzed/xsdk/aws_demos`. Escolha OK e, em seguida, selecione OK para salvar a variável e feche as Preferences (Preferências).

Compilação do projeto de demonstração do FreeRTOS

1. No IDE do XSDK, no menu, escolha Project (Projeto) e, em seguida, selecione Clean (Limpar).
2. Em Clean (Limpar), mantenha as opções com os valores padrão e escolha OK. O XSDK limpa e compilar todos os projetos e, em seguida, gera arquivos `.elf`.

> **Note**
>
> Para compilar todos os projetos sem limpá-los, escolha Project (Projeto) e, em seguida, selecione Build All (Compilar todos).
> Para compilar projetos individuais, selecione o projeto que deseja compilar, escolha Project (Projeto) e selecione Build Project (Compilar projeto).

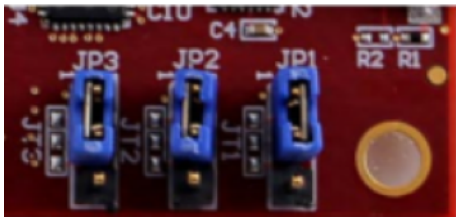
Geração de imagem de inicialização do projeto de demonstração do FreeRTOS

1. No IDE do XSDK, clique com o botão direito do mouse em aws_demos e escolha Create Boot Image (Criar imagem de inicialização).

2. Em Create Boot Image (Criar imagem de inicialização), escolha Create new BIF file (Criar novo arquivo BIF).
3. Ao lado de Output BIF file path (Caminho do arquivo BIF de saída), escolha Browse (Procurar) e selecione o `aws_demos.bif` localizado em *<freertos>*`/vendors/xilinx/microzed/aws_demos/aws_demos.bif`.
4. Escolha Adicionar.
5. Em Add new boot image partition (Adicionar nova partição da imagem de inicialização), ao lado de File path (Caminho do arquivo), escolha Browse (Navegar) e, em seguida, selecione `fsbl.elf` localizado em `vendors/xilinx/fsbl/Debug/fsbl.elf`.
6. Para o Partition type (Tipo de partição), escolha bootloader e selecione OK.
7. Em Create Boot Image (Criar imagem de inicialização), escolha Create Image (Criar imagem). Em Override Files (Substituir arquivos), escolha OK para substituir o `aws_demos.bif` existente e gerar o arquivo `BOOT.bin` em `projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin`.

Depuração do JTAG

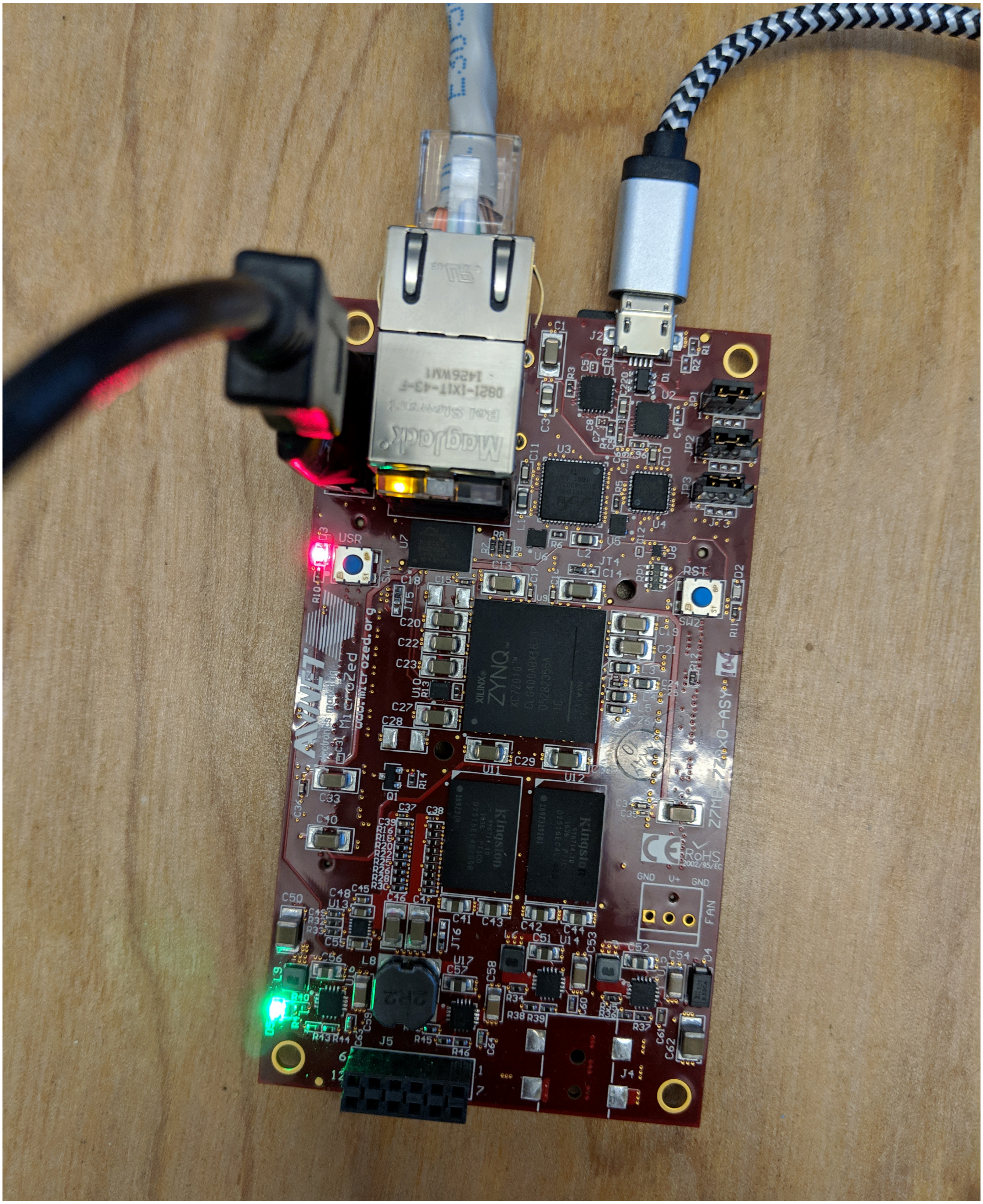
1. Configure os jumpers do modo de inicialização da sua MicroZed placa para o modo de inicialização JTAG.



2. Insira o cartão MicroSD no slot para cartão MicroSD localizado embaixo da porta USB-UART.

> **Note**
>
> Antes de depurar, faça backup de qualquer conteúdo existente no cartão MicroSD.

A placa deverá ser semelhante à seguinte:


USR
RST
GND
V+
GND
FAN
J5
J4

3. No IDE do XSDK, clique com o botão direito do mouse em aws_demos, escolha Debug As (Depurar como) e, em seguida, selecione 1 Launch on System Hardware (System Debugger) (1 Iniciar no hardware do sistema (depurador do sistema)).
4. Quando o depurador parar no ponto de interrupção em `main()`, no menu, escolha Run (Executar) e, em seguida, selecione Resume (Continuar).

> **Note**
>
> Na primeira vez que você executar a aplicação, um novo par de chave-certificado será importado para a memória não volátil. Para execuções subsequentes, você pode comentar `vDevModeKeyProvisioning()` no arquivo `main.c` antes de compilar novamente as imagens e o arquivo `BOOT.bin`. Isso impede a cópia dos certificados e da chave para armazenamento em cada execução.

Você pode optar por inicializar sua MicroZed placa a partir de um cartão microSD ou do flash QSPI para executar o projeto de demonstração do FreeRTOS. Para obter instruções, consulte Geração de imagem de inicialização do projeto de demonstração do FreeRTOS e Execução do projeto de demonstração do FreeRTOS.

Execução do projeto de demonstração do FreeRTOS

Para executar o projeto de demonstração do FreeRTOS, você pode inicializar MicroZed sua placa a partir de um cartão microSD ou do flash QSPI.

Ao configurar sua MicroZed placa para executar o projeto de demonstração do FreeRTOS, consulte o diagrama em. Configurar o MicroZed hardware Verifique se você conectou sua MicroZed placa ao computador.

Inicialização do projeto do FreeRTOS a partir de um cartão MicroSD

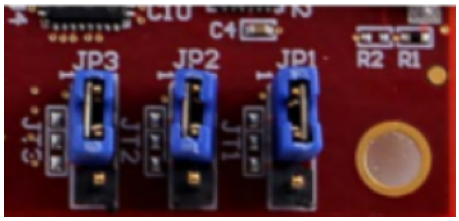
Formate o cartão microSD fornecido com o kit de IoT industrial Xilinx MicroZed.

1. Copie o arquivo `BOOT.bin` para o cartão MicroSD.
2. Insira o cartão no slot para cartão MicroSD embaixo da porta USB-UART.
3. Defina os jumpers do modo de MicroZed inicialização para o modo de inicialização SD.

4. Pressione o botão RST para redefinir o dispositivo e começar a inicialização da aplicação. Você também pode desconectar o cabo USB-UART da porta USB-UART e, em seguida, inserir o cabo novamente.

Inicialização do projeto de demonstração do FreeRTOS a partir de QSPI flash

1. Configure os jumpers do modo de inicialização da sua MicroZed placa para o modo de inicialização JTAG.



2. Verifique se o computador está conectado às portas USB-UART e JTAG Access. A luz Power Good LED verde deve estar iluminada.
3. No IDE do XSDK, no menu, escolha Xilinx e, em seguida, selecione Program Flash (Programar flash).
4. Em Program Flash Memory (Programar memória flash), a plataforma do hardware deverá ser preenchida automaticamente. Em Conexão, escolha seu servidor de MicroZed hardware para conectar sua placa ao computador host.

> **Note**
>
> Se estiver usando o cabo Xilinx Smart Lync JTAG, é necessário criar um servidor de hardware no IDE do XSDK. Escolha New (Novo) e defina o servidor.

5. Em Image File (Arquivo de imagem), insira o caminho de diretório para o arquivo de imagem `BOOT.bin`. Em vez disso, escolha Browse (Navegar) para navegar até o arquivo.
6. Em Offset (Deslocamento), insira **`0x0`**.
7. Em FSBL File (Arquivo FSBL), insira o caminho de diretório para o arquivo `fsbl.elf`. Em vez disso, escolha Browse (Navegar) para navegar até o arquivo.

8. Escolha Program (Programar) para programar a placa.
9. Após a conclusão da programação QSPI, remova o cabo USB-UART para desligar a placa.
10. Configure os jumpers do modo de inicialização da sua MicroZed placa para o modo de inicialização QSPI.
11. Insira o cartão no slot para cartão MicroSD localizado embaixo da porta USB-UART.

> **Note**
>
> Lembre-se de fazer backup de qualquer conteúdo existente no cartão MicroSD.

12. Pressione o botão RST para redefinir o dispositivo e começar a inicialização da aplicação. Você também pode desconectar o cabo USB-UART da porta USB-UART e, em seguida, inserir o cabo novamente.

Solução de problemas

Se você encontrar erros de compilação relacionados a caminhos incorretos, tente limpar e compilar o projeto novamente, conforme descrito em Compilação do projeto de demonstração do FreeRTOS.

Se estiver usando o Windows, lembre-se de usar barras ao definir as variáveis de substituição de strings no IDE do Windows XSDK.

Para obter mais informações sobre soluções de problemas gerais sobre os Conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Próximas etapas com o FreeRTOS

> **Important**
>
> Essa página se refere ao repositório do Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Depois de compilar, instalar e executar o projeto de demonstração do FreeRTOS para sua placa, você pode visitar o site FreeRTOS.org para saber mais sobre a Criação de um projeto do FreeRTOS. Lá também existem demonstrações de muitas bibliotecas do FreeRTOS que mostram como realizar

tarefas importantes, interagir com serviços do AWS IoT e programar recursos específicos da placa (como modems celulares). Para obter mais informações, consulte Categorias de bibliotecas do FreeRTOS.

O site FreeRTOS.org também tem informações detalhadas sobre o Kernel do FreeRTOS, bem como conceitos fundamentais do sistema operacional em tempo real. Para obter mais informações, consulte as páginas Documentação do desenvolvedor Kernel do FreeRTOS e Documentação secundária Kernel do FreeRTOS.

Atualizações sem fios do FreeRTOS

> **Note**
>
> Consulte AWS IoT sem fios no site do FreeRTOS para obter informações recentes sobre como realizar atualizações sem fios.

As atualizações over-the-air (OTA) permitem que você implante atualizações de firmware em um ou mais dispositivos da frota. Embora as atualizações OTA tenham sido projetadas para atualizar o firmware do dispositivo, você pode usá-las para enviar quaisquer arquivos para um ou mais dispositivos registrados na AWS IoT. Quando você envia atualizações OTA, recomendamos que você as assine digitalmente para que os dispositivos que recebem os arquivos possam verificar se não foram adulteradas no caminho.

Você pode usar o Code Signing para AWS IoT para assinar os arquivos ou pode assiná-los com suas próprias ferramentas de assinatura de código.

Ao criar uma atualização OTA, o Serviço do gerenciador de atualização OTA cria um trabalho do AWS IoT para notificar os dispositivos de que uma atualização está disponível. O aplicativo de demonstração OTA é executado no dispositivo e cria uma tarefa do que assina tópicos de notificação para trabalhos do AWS IoT e detecta mensagens de atualização. Quando uma atualização está disponível, o agente OTA publica solicitações no AWS IoT e recebe atualizações usando o protocolo HTTP ou MQTT, dependendo das configurações escolhidas. O agente OTA verifica a assinatura digital dos arquivos obtidos por download e, se for válida, instala a atualização do firmware. Se você não estiver usando o aplicativo de demonstração de atualização OTA do FreeRTOS, deverá integrar o Biblioteca sem fios do AWS IoT ao próprio aplicativo para obter o recurso de atualização de firmware.

As atualizações sem fios do FreeRTOS permitem:

- Assinar digitalmente o firmware antes da implantação.
- Implante novas imagens de firmware em um único dispositivo, um grupo de dispositivos ou toda a sua frota.
- Implante firmware em dispositivos à medida que eles são adicionados a grupos, redefinidos ou provisionados novamente.
- Verifique a autenticidade e a integridade do novo firmware depois de implantá-lo nos dispositivos.
- Monitore o progresso de uma implantação.
- Depure uma implantação com falha.

Marcação de recursos de OTA

Para ajudá-lo a gerenciar recursos OTA, é possível atribuir seus próprios metadados a atualizações e fluxos na forma de tags. As tags possibilitam a você categorizar seus recursos da AWS IoT de diferentes formas (como por finalidade, por proprietário ou por ambiente). Isso é útil quando você tem muitos recursos do mesmo tipo. Você pode identificar rapidamente um recurso com base nas tags que você atribuiu a ele.

Para obter mais informações, consulte Marcar seus recursos do AWS IoT.

Pré-requisitos de atualização do OTA

Para usar atualizações over-the-air (OTA), faça o seguinte:

- Verifique o Pré-requisitos para atualizações de OTA usando HTTP ou o Pré-requisitos para atualizações de OTA usando MQTT.
- Criação de um bucket do Amazon S3 para armazenar a atualização.
- Criação de uma função de serviço de atualização de OTA.
- Criação de uma política de usuário de OTA.
- Criação de um certificado de assinatura de código.
- Se você estiver usando o Code Signing para AWS IoT, Conceder acesso ao Code Signing para AWS IoT.
- Faça download do FreeRTOS com a biblioteca de OTA.

Criação de um bucket do Amazon S3 para armazenar a atualização

Os arquivos de atualização OTA são armazenados em buckets do Amazon S3.

Se você estiver usando o Code Signing para AWS IoT, o comando usado para criar um trabalho de assinatura de código utilizará um bucket de origem (onde a imagem de firmware não assinada está localizada) e um bucket de destino (onde a imagem de firmware assinada é gravada). Você pode especificar o mesmo bucket para a origem e o destino. Os nomes de arquivo são alterados para GUIDs para que os arquivos originais não sejam substituídos.

Como criar um bucket do Amazon S3

1. Faça login no Amazon S3 em https://console.aws.amazon.com/s3/.
2. Selecione Create bucket (Criar bucket).
3. Insira um nome para o bucket.
4. Em Configurações do bucket para Bloquear acesso público, mantenha Bloquear todo o acesso público selecionado para aceitar as permissões padrão.
5. Em versionamento de bucket, selecione Habilitar para manter todas as versões no mesmo bucket.
6. Selecione Create bucket (Criar bucket).

Para obter mais informações sobre o Amazon S3, consulte o Guia do usuário do Amazon Simple Storage Service.

Criação de uma função de serviço de atualização de OTA

O serviço de atualização OTA assume essa função para criar e gerenciar trabalhos de atualização OTA em seu nome.

Para criar uma função de serviço OTA

1. Faça login em https://console.aws.amazon.com/iam/.
2. No painel de navegação, escolha Roles (Funções).
3. Selecione Create role (Criar função).
4. Em Select type of trusted entity (Selecionar tipo de entidade confiável), selecione AWS Serviço.
5. Escolha IoT na lista de serviços da AWS.
6. Em Select your use case (Selecione seu caso de uso), escolha IoT.
7. Escolha Next: Permissions (Próximo: Permissões).
8. Escolha Next: Tags (Próximo: tags).

9. Escolha Next: Review (Próximo: revisar).
10. Digite um nome e uma descrição de função e escolha Create role (Criar função).

Para obter mais informações sobre perfis do IAM, consulte Perfis do IAM.

> **Important**
>
> Para resolver o problema de segurança de substituição confusa, você deve seguir as instruções do Guia do AWS IoT Core.

Para adicionar permissões de atualização OTA à função de serviço OTA

1. Na caixa de pesquisa na página do console do IAM, insira o nome do perfil e escolha-o na lista.
2. Escolha Attach policies (Anexar políticas).
3. Na caixa Search (Pesquisar), insira "AmazonFreeRTOSOTAUpdate", selecione AmazonFreeRTOSOTAUpdate na lista de políticas filtradas e escolha Attach policy (Anexar política) para anexar a política à função de serviço.

Para adicionar as permissões necessárias do IAM ao perfil de serviço OTA

1. Na caixa de pesquisa na página do console do IAM, insira o nome do perfil e escolha-o na lista.
2. Escolha Add inline policy (Adicionar política em linha).
3. Escolha a guia JSON.
4. Copie e cole o documento de política a seguir na caixa 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"
      }
```

```
    ]
}
```

Certifique-se de substituir *your_account_id* pelo ID da conta da AWS e *your_role_name* pelo nome da função de serviço OTA.

5. Escolha Review policy (Revisar política).
6. Insira um nome para a política e escolha Create policy (Criar política).

> **Note**
>
> O procedimento a seguir não será necessário se o nome do bucket do Amazon S3 começar com "afr-ota". Se isso acontecer, a política gerenciada da AWS `AmazonFreeRTOSOTAUpdate` já inclui as permissões necessárias.

Para adicionar as permissões necessárias do Amazon S3 ao perfil de serviço OTA

1. Na caixa de pesquisa na página do console do IAM, insira o nome do perfil e escolha-o na lista.
2. Escolha Add inline policy (Adicionar política em linha).
3. Escolha a guia JSON.
4. Copie e cole o documento de política a seguir na caixa.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
            ]
        }
    ]
}
```

Essa política concede ao perfil de serviço OTA permissão para ler objetos do Amazon S3. Certifique-se de substituir *example-bucket* pelo nome do bucket.

5. Escolha Review policy (Revisar política).
6. Insira um nome para a política e escolha Create policy (Criar política).

Criação de uma política de usuário de OTA

Você deve conceder ao usuário do permissão para executar atualizações over-the-air. O usuário do deve ter permissões para:

- Acessar o bucket do S3 onde suas atualizações de firmware estão armazenadas.
- Acessar os certificados armazenados no AWS Certificate Manager.
- Acessar o atributo de entrega de arquivos baseado em MQTT do AWS IoT.
- Acessar as atualizações OTA do FreeRTOS.
- Acessar os trabalhos da AWS IoT.
- Acessar o IAM.
- Acesse o Code Signing para AWS IoT. Consulte Conceder acesso ao Code Signing para AWS IoT.
- Listar as plataformas de hardware FreeRTOS.
- Marcar e desmarcar recursos do AWS IoT.

Para conceder as permissões necessárias ao seu usuário, consulte política do IAM. Consulte também, Como autorizar usuários e serviços em nuvem a usar os Trabalhos do AWS IoT.

Para fornecer o acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos no AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Create a permission set (Criação de um conjunto de permissões) no Guia do usuário do AWS IAM Identity Center.
- Usuários gerenciados no IAM usando um provedor de identidades:

  Crie um perfil para a federação de identidades. Siga as instruções em Criar um perfil para um provedor de identidades de terceiros (federação) no Guia do usuário do IAM.
- Usuários do IAM:

- Crie um perfil que seu usuário possa assumir. Siga as instruções em Creating a role for an IAM user (Criação de um perfil para um usuário do IAM) no Guia do usuário do IAM.
- (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Criação de um certificado de assinatura de código

Para assinar digitalmente imagens de firmware, você precisa de um certificado de assinatura de código e uma chave privada. Para fins de teste, você pode criar um certificado autoassinado e uma chave privada. Para ambientes de produção, adquira um certificado por meio de uma autoridade de certificação (CA) reconhecida.

Plataformas diferentes exigem tipos diferentes de certificados de assinatura de código. A seção a seguir descreve como criar certificados de assinatura de código para diferentes plataformas qualificadas para o FreeRTOS.

Tópicos

- Criação de um certificado de assinatura de código para o Texas Instruments CC3220SF-LAUNCHXL
- Criação de um certificado de assinatura de código para o Espressif ESP32
- Criação de um certificado de assinatura de código para Nordic nrf52840-dk
- Criação de um certificado de assinatura de código para o simulador do Windows do FreeRTOS
- Criação de um certificado de assinatura de código para o hardware personalizado

Criação de um certificado de assinatura de código para o Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

O Kit de desenvolvimento SimpleLink Wi-Fi CC3220SF Wireless Microcontroller Launchpad oferece suporte a duas cadeias de certificados para assinatura de código de firmware:

- Produção (certificado-catálogo)

  Para usar a cadeia de certificados de produção, você deve adquirir um certificado de assinatura de código comercial e usar a Ferramenta TI Uniflash para definir a placa para o modo de produção.

- Teste e desenvolvimento (certificado-playground)

  A cadeia de certificados playground permite experimentar as atualizações OTA com um certificado de assinatura de código autoassinado.

Use a AWS Command Line Interface para importar o certificado de assinatura de código, a chave privada e a cadeia de certificados no AWS Certificate Manager. Para obter mais informações, consulte Instalar a AWS CLI no Guia do usuário da AWS Command Line Interface.

Faça download e instale a versão mais recente do SDK SimpleLink CC3220. Por padrão, os arquivos necessários estão localizados aqui:

`C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground` (Windows)

`/Applications/Ti/simplelink_cc32xx_version/tools/cc32xx_tools/certificate-playground` (macOS)

Os certificados no SimpleLink CC3220 SDK estão no formato DER. Para criar um certificado de assinatura de código autoassinado, você deve convertê-lo para o formato PEM.

Siga estas etapas para criar um certificado de assinatura de código que esteja vinculado à hierarquia de certificados playground da Texas Instruments e atenda aos critérios do AWS Certificate Manager e do Code Signing para AWS IoT.

> **Note**
>
> Para criar um certificado de assinatura de código, é necessário instalar o OpenSSL na máquina. Depois de instalar o OpenSSL, verifique se `openssl` está atribuído ao executável do OpenSSL no prompt de comando ou ambiente de terminal.

Como criar um certificado de assinatura de código autoassinado

1. Abra um terminal ou prompt de comando com permissões de administrador.
2. No diretório de trabalho, use o texto a seguir para criar um arquivo chamado `cert_config.txt`. Substitua *`test_signer@amazon.com`* pelo seu endereço de e-mail.

```
[ req ]
prompt             = no
distinguished_name = my dn

[ my dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

3. Crie uma chave privada e uma solicitação de assinatura 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. Converta a chave privada da CA raiz do playground do Texas Instruments do formato DER para o formato PEM.

   A chave privada da CA raiz do playground do TI está localizada aqui:

   `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. Converta o certificado da CA raiz do playground do Texas Instruments do formato DER para o formato PEM.

   O certificado raiz do playground do TI está localizada aqui:

`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. Assine o CSR com a CA raiz do 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. Converta seu código de assinatura de certificado (`tisigner.crt.pem`) para o formato DER:

```
openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
```

> **Note**
>
> Você escreve o certificado `tisigner.crt.der` na placa de desenvolvimento do TI mais tarde.

8. Importe o certificado de assinatura de código, a chave privada e a cadeia de certificados no 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 exibe um ARN para o seu certificado. Você precisa deste ARN quando cria um trabalho de atualização OTA.

> **Note**
>
> Essa etapa foi escrita presumindo que você usará o Code Signing para AWS IoT para assinar as imagens de firmware. Embora o uso do Code Signing para AWS IoT seja recomendado, você pode assinar as imagens de firmware manualmente.

Criação de um certificado de assinatura de código para o Espressif ESP32

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

As placas Espressif ESP32 oferecem suporte a um SHA-256 autoassinado com certificado de assinatura de código ECDSA.

> **Note**
>
> Para criar um certificado de assinatura de código, é necessário instalar o OpenSSL na máquina. Depois de instalar o OpenSSL, verifique se `openssl` está atribuído ao executável do OpenSSL no prompt de comando ou ambiente de terminal.
> Use a AWS Command Line Interface para importar o certificado de assinatura de código, a chave privada e a cadeia de certificados no AWS Certificate Manager. Para obter informações sobre como instalar a AWS CLI, consulte Instalar a AWS CLI.

1. No diretório de trabalho, use o texto a seguir para criar um arquivo chamado `cert_config.txt`. Substitua *`test_signer@amazon.com`* pelo seu endereço de e-mail:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Crie uma chave privada de assinatura 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. Crie um certificado de assinatura 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 o certificado de assinatura de código, a chave privada e a cadeia de certificados no AWS Certificate Manager:

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Este comando exibe um ARN para o seu certificado. Você precisa deste ARN quando cria um trabalho de atualização OTA.

> **Note**
>
> Essa etapa foi escrita presumindo que você usará o Code Signing para AWS IoT para assinar as imagens de firmware. Embora o uso do Code Signing para AWS IoT seja recomendado, você pode assinar as imagens de firmware manualmente.

Criação de um certificado de assinatura de código para Nordic nrf52840-dk

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

O Nordic nrf52840-dk oferece suporte a um SHA256 autoassinado com certificado de assinatura de código ECDSA.

> **Note**
>
> Para criar um certificado de assinatura de código, instale o OpenSSL na máquina. Depois de instalar o OpenSSL, verifique se `openssl` está atribuído ao executável do OpenSSL no prompt de comando ou ambiente de terminal.
> Use a AWS Command Line Interface para importar o certificado de assinatura de código, a chave privada e a cadeia de certificados no AWS Certificate Manager. Para obter informações sobre como instalar a AWS CLI, consulte Instalar a AWS CLI.

1. No diretório de trabalho, use o texto a seguir para criar um arquivo chamado `cert_config.txt`. Substitua *`test_signer@amazon.com`* pelo seu endereço de e-mail:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Crie uma chave privada de assinatura 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. Crie um certificado de assinatura 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 o certificado de assinatura de código, a chave privada e a cadeia de certificados no AWS Certificate Manager:

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Este comando exibe um ARN para o seu certificado. Você precisa deste ARN quando cria um trabalho de atualização OTA.

> **Note**
>
> Essa etapa foi escrita presumindo que você usará o Code Signing para AWS IoT para assinar as imagens de firmware. Embora o uso do Code Signing para AWS IoT seja recomendado, você pode assinar as imagens de firmware manualmente.

Criação de um certificado de assinatura de código para o simulador do Windows do FreeRTOS

O simulador do Windows do FreeRTOS exige um certificado de assinatura de código com uma chave ECDSA P-256 e o hash SHA-256 para executar atualizações OTA. Se você não tiver um certificado de assinatura de código, siga estas etapas para criar um:

> **Note**
>
> Para criar um certificado de assinatura de código, é necessário instalar o OpenSSL na máquina. Depois de instalar o OpenSSL, verifique se `openssl` está atribuído ao executável do OpenSSL no prompt de comando ou ambiente de terminal.
> Use a AWS Command Line Interface para importar o certificado de assinatura de código, a chave privada e a cadeia de certificados no AWS Certificate Manager. Para obter informações sobre como instalar a AWS CLI, consulte Instalar a AWS CLI.

1. No diretório de trabalho, use o texto a seguir para criar um arquivo chamado `cert_config.txt`. Substitua *`test_signer@amazon.com`* pelo seu endereço de e-mail:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
```

```
extendedKeyUsage = codeSigning
```

2. Crie uma chave privada de assinatura 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. Crie um certificado de assinatura 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 o certificado de assinatura de código, a chave privada e a cadeia de certificados no AWS Certificate Manager:

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

   Este comando exibe um ARN para o seu certificado. Você precisa deste ARN quando cria um trabalho de atualização OTA.

   > **Note**
   >
   > Essa etapa foi escrita presumindo que você usará o Code Signing para AWS IoT para assinar as imagens de firmware. Embora o uso do Code Signing para AWS IoT seja recomendado, você pode assinar as imagens de firmware manualmente.

Criação de um certificado de assinatura de código para o hardware personalizado

Usando um conjunto de ferramentas apropriado, crie um certificado autoassinado e uma chave privada para o hardware.

Use a AWS Command Line Interface para importar o certificado de assinatura de código, a chave privada e a cadeia de certificados no AWS Certificate Manager. Para obter informações sobre como instalar a AWS CLI, consulte Instalar a AWS CLI.

Depois de criar o certificado de assinatura de código, você pode usar a AWS CLI para importá-lo para o ACM:

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://
code-sign.key
```

A saída desse comando exibe um ARN para o seu certificado. Você precisa deste ARN quando cria um trabalho de atualização OTA.

O ACM exige certificados para usar algoritmos e tamanhos de chaves específicos. Para obter mais informações, consulte Pré-requisitos para importar certificados. Para obter mais informações sobre o ACM, consulte Importação de certificados para o AWS Certificate Manager.

Você deve copiar, colar e formatar o conteúdo do certificado de assinatura de código no arquivo `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` que faz parte do código do FreeRTOS que você obterá por download posteriormente.

Conceder acesso ao Code Signing para AWS IoT

Para fornecer o acesso, adicione as permissões aos seus usuários, grupos ou perfis:

- Usuários e grupos no AWS IAM Identity Center:

  Crie um conjunto de permissões. Siga as instruções em Create a permission set (Criação de um conjunto de permissões) no Guia do usuário do AWS IAM Identity Center.
- Usuários gerenciados no IAM usando um provedor de identidades:

  Crie um perfil para a federação de identidades. Siga as instruções em Criar um perfil para um provedor de identidades de terceiros (federação) no Guia do usuário do IAM.
- Usuários do IAM:
  - Crie um perfil que seu usuário possa assumir. Siga as instruções em Creating a role for an IAM user (Criação de um perfil para um usuário do IAM) no Guia do usuário do IAM.
  - (Não recomendado) Vincule uma política diretamente a um usuário ou adicione um usuário a um grupo de usuários. Siga as instruções em Adição de permissões a um usuário (console) no Guia do usuário do IAM.

Faça download do FreeRTOS com a biblioteca de OTA

Você pode clonar ou fazer download do FreeRTOS do GitHub. Consulte o arquivo README.md para obter instruções.

Para obter informações sobre como configurar e executar o aplicativo de demonstração OTA, consulte Aplicativo de demonstração de atualizações remotas.

⚠ Important

- Neste tópico, o caminho para o diretório de download do FreeRTOS é chamado de *freertos*.
- Caracteres de espaço no caminho *freertos* podem causar falhas na compilação. Ao clonar ou copiar o repositório, verifique se o caminho criado não contém caracteres de espaço.
- O tamanho máximo de um caminho de arquivo no Microsoft Windows é 260 caracteres. Caminhos longos de diretório de download do FreeRTOS podem causar falhas de compilação.
- O código-fonte pode conter links simbólicos, por isso se estiver usando o Windows para extrair o arquivo, talvez seja necessário:
  - Habilitar o modo Desenvolvedor ou,
  - Usar um console com privilégios de administrador.

  Dessa forma, o Windows pode criar links simbólicos adequadamente ao extrair o arquivamento. Caso contrário, os links simbólicos serão gravados como arquivos comuns que contêm os caminhos dos links simbólicos como texto ou estão vazios. Para obter mais informações, consulte a entrada no blog Symlinks no Windows 10!.

  Se você usa o Git no Windows, você deve habilitar o modo Desenvolvedor ou deve:
  - Definir `core.symlinks` como verdadeiro com o seguinte comando:

    ```
    git config --global core.symlinks true
    ```
  - Use um console com privilégios de administrador sempre que usar um comando git que grava no sistema (por exemplo, git pull, git clone, e git submodule update --init --recursive).

Pré-requisitos para atualizações de OTA usando MQTT

Esta seção descreve os requisitos gerais para o uso de MQTT para executar atualizações de OTA (Over-The-Air).

Requisitos mínimos

- O firmware do dispositivo deve incluir as bibliotecas do FreeRTOS necessárias (coreMQTT Agent, atualização OTA e suas dependências).
- A versão 1.4.0 ou posterior do FreeRTOS é necessária. Recomendamos usar a versão mais recente sempre que possível.

Configurações

A partir da versão 201912.00, o OTA do FreeRTOS pode usar o protocolo HTTP ou MQTT para transferir imagens de atualização de firmware do AWS IoT para dispositivos. Se você especificar os dois protocolos ao criar uma atualização OTA no FreeRTOS, cada dispositivo determinará o protocolo usado para transferir a imagem. Consulte Pré-requisitos para atualizações de OTA usando HTTP para obter mais informações.

Por padrão, a configuração de protocolos OTA em `ota_config.h` é usar o protocolo MQTT.

Configurações específicas do dispositivo

Nenhum.

Uso de memória

Quando o MQTT for usado para transferência de dados, não será necessária nenhuma memória adicional para a conexão MQTT, pois ela é compartilhada entre operações de controle e de dados.

Política de dispositivo

Cada dispositivo que recebe uma atualização OTA usando MQTT deve ser registrado como uma coisa dentro do AWS IoT, e a coisa deve ter uma política anexada como a listada aqui. Você pode encontrar mais informações sobre os itens nos objetos `"Resource"` e `"Action"` em Ações da política principal do AWS IoT e Recursos da ação principal do 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/*"
            ]
        }
    ]
}
```

Observações

- As permissões do `iot:Connect` permitem que o dispositivo seja conectado ao AWS IoT pelo MQTT.
- As permissões de `iot:Subscribe` e `iot:Publish` para tópicos de trabalhos da AWS IoT (`.../jobs/*`) permitem que o dispositivo conectado receba notificações de trabalho e documentos de trabalho, e publique o estado de conclusão da execução de um trabalho.
- As permissões de `iot:Subscribe` e `iot:Publish` para tópicos de fluxos do OTA de AWS IoT (`.../streams/*`) permitem que o dispositivo conectado obtenha dados de atualização do OTA da AWS IoT. Essas permissões são necessárias para executar atualizações de firmware pelo MQTT.
- As permissões de `iot:Receive` permitem que o AWS IoT Core publique mensagens sobre esses tópicos no dispositivo conectado. Essa permissão é verificada em cada entrega de uma

mensagem de MQTT. Você pode usar essa permissão para revogar o acesso a clientes que estão inscritos em um tópico atualmente.

Pré-requisitos para atualizações de OTA usando HTTP

Esta seção descreve os requisitos gerais para usar HTTP para executar atualizações over-the-air (OTA). A partir da versão 201912.00, o OTA do FreeRTOS pode usar o protocolo HTTP ou MQTT para transferir imagens de atualização de firmware do AWS IoT para dispositivos.

> **Note**
>
> - Embora o protocolo HTTP possa ser usado para transferir a imagem do firmware, a biblioteca coreMQTT Agent ainda é necessária porque outras interações com o AWS IoT Core usam a biblioteca coreMQTT Agent, incluindo o envio ou recebimento de notificações de execução de tarefas, documentos de trabalho e atualizações de status de execução.
> - Ao especificar os protocolos MQTT e HTTP para o trabalho de atualização de OTA, a configuração do software do agente OTA em cada dispositivo individual determina o protocolo usado para transferir a imagem de firmware. Para alterar o agente OTA do método de protocolo MQTT padrão para o protocolo HTTP, modifique os arquivos de cabeçalho usados para compilar o código-fonte do FreeRTOS para o dispositivo.

Requisitos mínimos

- O firmware do dispositivo deve incluir as bibliotecas do FreeRTOS necessárias (agente coreMQTT, HTTP, agente OTA e suas dependências).
- A versão 201912.00 ou posterior o FreeRTOS é necessária para alterar a configuração de protocolos OTA a fim de habilitar a transferência de dados OTA via HTTP.

Configurações

Consulte a seguinte configuração dos protocolos OTA no arquivo `\vendors\boards\board\aws_demos\config_files\ota_config.h`.

Para habilitar a transferência de dados OTA via HTTP

1. Mude `configENABLED_DATA_PROTOCOLS` para `OTA_DATA_OVER_HTTP`.

2. Quando o OTA é atualizado, você pode especificar ambos os protocolos para que o protocolo MQTT ou HTTP possa ser usado. Você pode definir o protocolo primário usado pelo dispositivo para HTTP alterando `configOTA_PRIMARY_DATA_PROTOCOL` para `OTA_DATA_OVER_HTTP`.

> **Note**
>
> HTTP só é compatível com operações de dados OTA. Para operações de controle, você deve usar MQTT.

Configurações específicas do dispositivo

ESP32

Devido a uma quantidade limitada de RAM, você deve desativar o BLE quando habilitar HTTP como protocolo de dados OTA. No arquivo [vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h](), altere `configENABLED_NETWORKS` para `AWSIOT_NETWORK_TYPE_WIFI` somente.

```
/**
     * @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 memória

Quando MQTT for usado para transferência de dados, não será necessária nenhuma memória de heap adicional para a conexão MQTT, pois ela é compartilhada entre controle e operações de dados. No entanto, é necessário ter memória de heap adicional para habilitar dados via HTTP. A seguir

estão os dados de uso de memória de heap para todas as plataformas compatíveis, calculados usando a API `xPortGetFreeHeapSize` do FreeRTOS. Assegure-se de que há RAM suficiente para usar a biblioteca OTA.

Texas Instruments CC3220SF-LAUNCHXL

Operações de controle (MQTT): 12 KB

Operações de dados (HTTP): 10 KB

> **Note**
>
> TI usa significativamente menos RAM porque faz SSL em hardware, então não usa a biblioteca mbedtls.

Microchip Curiosity PIC32MZEF

Operações de controle (MQTT): 65 KB

Operações de dados (HTTP): 43 KB

Espressif ESP32

Operações de controle (MQTT): 65 KB

Operações de dados (HTTP): 45 KB

> **Note**
>
> BLE em ESP32 utiliza cerca de 87 KB RAM. Não há RAM suficiente para habilitar todos eles, o que é mencionado nas configurações específicas do dispositivo acima.

Windows Simulator

Operações de controle (MQTT): 82 KB

Operações de dados (HTTP): 63 KB

Nordic nrf52840-dk

Não há suporte para HTTP.

Política de dispositivo

Esta política permite usar MQTT ou HTTP para atualizações de OTA.

Cada dispositivo que recebe uma atualização OTA usando HTTP deve ser registrado como uma coisa no AWS IoT, e a coisa deve ter uma política anexada, como a listada aqui. Você pode encontrar mais informações sobre os itens nos objetos `"Resource"` e `"Action"` em Ações da política principal do AWS IoT e Recursos da ação principal do 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/*"
            ]
        }
    ]
}
```

Observações

- As permissões do `iot:Connect` permitem que o dispositivo seja conectado ao AWS IoT pelo MQTT.
- As permissões de `iot:Subscribe` e `iot:Publish` para tópicos de trabalhos da AWS IoT (`.../jobs/*`) permitem que o dispositivo conectado receba notificações de trabalho e documentos de trabalho, e publique o estado de conclusão da execução de um trabalho.
- As permissões de `iot:Receive` permitem que o AWS IoT Core publique mensagens sobre esses tópicos no dispositivo conectado atualmente. Essa permissão é verificada em cada entrega de uma mensagem de MQTT. Você pode usar essa permissão para revogar o acesso a clientes que estão inscritos em um tópico atualmente.

Tutorial do OTA

Essa seção contém um tutorial para atualização de firmware em dispositivos que executam o FreeRTOS usando atualizações OTA. Além de imagens de firmware, você pode usar uma atualização OTA para enviar qualquer tipo de arquivo para um dispositivo conectado ao AWS IoT.

Você pode usar o console da AWS IoT ou a AWS CLI para criar uma atualização OTA. O console é a maneira mais fácil de começar com o OTA porque ele faz grande parte do trabalho para você. A AWS CLI é útil quando você está automatizando trabalhos de atualização OTA, trabalhando com um grande número de dispositivos ou usando dispositivos que não foram qualificados para o FreeRTOS. Para obter mais informações sobre os dispositivos qualificados para o FreeRTOS, consulte o site Parceiros do FreeRTOS.

Para criar uma atualização OTA

1. Implante uma versão inicial do firmware em um ou mais dispositivos.
2. Verifique se o firmware está funcionando corretamente.
3. Quando uma atualização de firmware for necessária, faça as alterações no código e crie a nova imagem.
4. Se você assinar manualmente o firmware, assine e envie a imagem de firmware assinada ao bucket do Amazon S3. Se você estiver usando o Code Signing para AWS IoT, faça upload da imagem de firmware não assinada em um bucket do Amazon S3.
5. Crie uma atualização OTA.

Ao criar uma atualização OTA, especifique o protocolo de entrega de imagem (MQTT ou HTTP) ou especifique ambos para permitir que o dispositivo escolha. O agente OTA do FreeRTOS no dispositivo recebe a imagem de firmware atualizada e verifica a assinatura digital, a soma de verificação e o número da versão da nova imagem. Se a atualização do firmware for verificada, o dispositivo será redefinido e, com base na lógica definida pelo aplicativo, confirmará a atualização. Se os dispositivos não estiverem executando o FreeRTOS, será necessário implementar um agente OTA que seja executado em seus dispositivos.

Instalação do firmware inicial

Para atualizar o firmware, é necessário instalar uma versão inicial do firmware que usa a biblioteca do agente OTA para detectar trabalhos de atualização do OTA. Se você não estiver executando o FreeRTOS, ignore essa etapa. Em vez disso, você precisa copiar sua implementação do agente OTA nos dispositivos.

Tópicos

- Instalação da versão inicial do firmware no Texas Instruments CC3220SF-LAUNCHXL
- Instalação da versão inicial do firmware no Espressif ESP32
- Instalação da versão inicial do firmware no Nordic nRF52840 DK
- Firmware inicial no simulador do Windows
- Instalação da versão inicial do firmware em uma placa personalizada

Instalação da versão inicial do firmware no Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Essas etapas foram escritas presumindo que você já criou o projeto `aws_demos`, conforme descrito em Fazer download, compilação, instalação e execução de demonstração OTA do FreeRTOS no CC3220SF-LAUNCHXL da Texas Instruments.

1. No Texas Instruments CC3220SF-LAUNCHXL, coloque o jumper SOP no conjunto de pinos do meio (posição = 1) e redefina a placa.
2. Faça download e instale a ferramenta TI Uniflash.
3. Inicie o Uniflash. Na lista de configurações, escolha CC3220SF-LAUNCHXL e escolha Start Image Creator (Iniciar criador de imagens).
4. Escolha New Project (Novo projeto).
5. Na página Start new project (Iniciar novo projeto), insira um nome para o projeto. Em Device Type (Tipo de dispositivo), selecione CC3220SF. Em Device Mode (Modo de dispositivo), escolha Develop (Desenvolver). Escolha Create Project (Criar projeto).
6. Desconecte o emulador de terminal.
7. No lado direito da janela do aplicativo Uniflash, escolha Connect (Conectar).
8. Em Advanced (Avançado), Files (Arquivos), selecione User Files (Arquivos de usuário).
9. No painel de seleção File (Arquivo), escolha o ícone Add File (Adicionar arquivo)

.
10. Navegue até o diretório `/Applications/Ti/simplelink_cc32xx_sdk_`*`version`*`/tools/cc32xx_tools/certificate-playground`, selecione `dummy-root-ca-cert`, Open (Abrir) e Write (Gravar).
11. No painel de seleção File (Arquivo), escolha o ícone Add File (Adicionar arquivo)

.
12. Navegue até o diretório de trabalho em que você criou o certificado de assinatura de código e a chave privada, selecione `tisigner.crt.der`, Open (Abrir) e, em seguida, Write (Gravar).
13. Na lista suspensa Action (Ação), escolha Select MCU Image (Selecionar imagem do MCU) e, em seguida, Browse (Navegar) para escolher a imagem de firmware para gravar no dispositivo (aws_demos.bin). Esse arquivo encontra-se no diretório *`freertos`*`/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug`. Escolha Abrir.
    a. Na caixa de diálogo de arquivo, confirme se o nome do arquivo está definido como `mcuflashimg.bin`.
    b. Marque a caixa de seleção Vendor (Fornecedor).
    c. Em File Token (Token de arquivo), digite **`1952007250`**.
    d. Em Private Key File Name (Nome do arquivo de chave privada), selecione Browse (Navegar) e `tisigner.key` no diretório de trabalho em que você criou o certificado de assinatura de código e a chave privada.

e. Em Certification File Name (Nome do arquivo de certificação), escolha `tisigner.crt.der`.

f. Selecione Write (Gravar).

14. No painel à esquerda, em Files (Arquivos), escolha Service Pack (Pacote de serviço).
15. Em Service Pack File Name (Nome do arquivo do pacote de serviço), selecione Browse (Navegar), navegue até `simplelink_cc32x_sdk_`***`version`***`/tools/cc32xx_tools/servicepack-cc3x20`, selecione `sp_3.7.0.1_2.0.0.0_2.2.0.6.bin` e Open (Abrir).
16. No painel à esquerda, em Files (Arquivos), escolha Trusted Root-Certificate Catalog (Catálogo de certificados raiz confiável).
17. Desmarque a caixa de seleção Use default Trusted Root-Certificate Catalog (Usar catálogo de certificados raiz confiável padrão).
18. Em Source File (Arquivo de origem), selecione Browse (Navegar), selecione simplelink_cc32xx_sdk_***version***/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst e selecione Open (Abrir).
19. Em Signature Source File (Arquivo de origem de assinatura), selecione Browse (Navegar), selecione simplelink_cc32xx_sdk_***version***/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst.signed_3220.bin e selecione Open (Abrir).
20. Selecione o botão

    para salvar o projeto.
21. Selecione o botão

22. Selecione Program Image (Create and Program) (Imagem do programa (criar e programar)).
23. Após a conclusão do processo de programação, coloque o jumper SOP no primeiro conjunto de pinos (posição = 0), reconfigure a placa e reconecte o emulador de terminal para garantir que a saída seja a mesma de quando você depurou a demonstração com o Code Composer Studio. Anote o número da versão do aplicativo na saída do terminal. Você usa esse número de versão posteriormente para verificar se seu firmware foi atualizado por uma atualização OTA.

    O terminal deve exibir a saída como a seguir.

```
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
```

Instalação da versão inicial do firmware no Espressif ESP32

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Esse guia foi escrito presumindo que você já realizou as etapas descritas em Conceitos básicos do Espressif ESP32-DevKitC e do ESP-WROVER-KIT e em Pré-requisitos de atualização Over-the-Air (OTA). Antes de tentar uma atualização OTA, é aconselhável tentar executar o projeto de

demonstração MQTT descrito em Conceitos básicos do FreeRTOS para garantir que a placa e a cadeia de ferramentas estejam configuradas corretamente.

Para atualizar uma imagem inicial de fábrica na 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 e defina CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED ou CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED.
2. Copie o certificado de assinatura de código SHA-256/ECDSA no formato PEM gerado no Pré-requisitos de atualização do OTA para o vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h. Ele deve ser formatado como a seguir.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE  \
"-----BEGIN CERTIFICATE-----\n" \
"...base64 data...\n" \
"-----END CERTIFICATE-----\n";
```

3. Com a demonstração da atualização OTA selecionada, siga as mesmas etapas descritas em Conceitos básicos do ESP32 para criar e atualizar a imagem. Se você criou e atualizou o projeto anteriormente, talvez seja necessário executar make clean primeiro. Depois de executar make flash monitor, você verá algo parecido com o exemplo a seguir. A ordem de algumas mensagens pode variar, porque o aplicativo de demonstração executa várias tarefas de uma só 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. A placa ESP32 agora está detectando as atualizações OTA. O monitor ESP-IDF é iniciado pelo comando `make flash monitor`. Você pode pressionar Ctrl+] para sair. Você também pode usar o seu programa de terminal TTY preferencial (por exemplo, PuTTY, Tera Term ou GNU Screen) para detectar a saída serial da placa. Lembre-se de que a conexão com a porta serial da placa pode causar a reinicialização.

Instalação da versão inicial do firmware no Nordic nRF52840 DK

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este guia está escrito com a suposição de que você já tenha executado as etapas em Conceitos básicos do Nordic nRF52840-DK e Pré-requisitos de atualização Over-the-Air (OTA). Antes de tentar uma atualização OTA, é aconselhável executar o projeto de demonstração MQTT descrito em Conceitos básicos do FreeRTOS para garantir que a placa e a cadeia de ferramentas estejam configuradas corretamente.

Para atualizar uma imagem inicial de fábrica na placa

1. Aberto *`freertos`*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h`.
2. Substitua `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` por `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Com a demonstração da atualização OTA selecionada, siga as mesmas etapas descritas em Conceitos básicos do Nordic nRF52840-DK para criar e atualizar a imagem.

Você deve ver saída semelhante ao seguinte:

```
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
```

A placa agora está detectando as atualizações OTA.

Firmware inicial no simulador do Windows

Quando você usa o simulador do Windows, não há necessidade de atualizar uma versão inicial do firmware. O simulador do Windows é parte do aplicativo `aws_demos`, que também inclui o firmware.

Instalação da versão inicial do firmware em uma placa personalizada

Usando seu IDE, crie o projeto `aws_demos`, certificando-se de incluir a biblioteca OTA. Para obter mais informações sobre a estrutura do código-fonte do FreeRTOS, consulte Demonstrações do FreeRTOS.

Inclua o certificado de assinatura de código, a chave privada e a cadeia de confiança do certificado no projeto do FreeRTOS ou no dispositivo.

Usando a ferramenta apropriada, grave o aplicativo na placa e verifique se ele está sendo executado corretamente.

Atualização da versão do firmware

O agente OTA incluído no FreeRTOS verifica a versão de qualquer atualização e a instala apenas se for mais recente que a versão de firmware existente. As etapas a seguir mostram como incrementar a versão do firmware do aplicativo de demonstração OTA.

1. Abra o projeto `aws_demos` no IDE.
2. Localize o arquivo `/vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` e incremente o valor de APP_VERSION_BUILD.
3. Para agendar uma atualização para uma plataforma rx65n da Renesas com um tipo de arquivo diferente de 0 (arquivos sem firmware), você deve assinar o arquivo com a ferramenta Renesas Secure Flash Programmer, caso contrário, ele falhará na verificação da assinatura no dispositivo. A ferramenta cria um pacote de arquivos assinado com a extensão `.rsu` que é um tipo de arquivo de propriedade da Renesas. A ferramenta pode ser encontrada no Github. É possível usar o seguinte comando de exemplo para gerar a imagem:

   ```
   "Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure
    Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
   ```
4. Recrie o projeto.

É necessário copiar a atualização de firmware no bucket do Amazon S3 criado, conforme descrito em Criação de um bucket do Amazon S3 para armazenar a atualização. O nome do arquivo que você precisa copiar no Amazon S3 depende da plataforma de hardware que você está usando:

- Texas Instruments CC3220SF-LAUNCHXL: `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- Espressif ESP32: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

Criação de uma atualização do OTA (console de AWS IoT)

1. No painel de navegação do console do AWS IoT, escolha Gerenciar selecione Ações remotas e, em seguida, escolha Trabalhos.
2. Escolha Create job (Criar trabalho).

3. Em Tipo de trabalho, selecione Criar trabalho de atualização OTA do FreeRTOS e escolha Próximo.
4. Em Propriedades do trabalho, insira um nome do trabalho e (opcionalmente) insira uma Descrição do trabalho e escolha Próximo.
5. Você pode implantar uma atualização OTA em um único dispositivo ou em um grupo de dispositivos. Em Dispositivos a serem atualizados, escolha um ou mais itens ou grupos de itens no menu suspenso.
6. Em Selecionar o protocolo para transferência de arquivos, escolha HTTP, MQTT ou ambos para permitir que cada dispositivo determine o protocolo a ser usado.
7. Em Assinar e escolher o arquivo, selecione Assinar um novo arquivo para mim.
8. Em Perfil de assinatura de código, escolha Criar perfil.
9. Em Create a code signing profile (Criar um perfil de assinatura de código), insira um nome para o perfil de assinatura de código.

    a. Em Device hardware platform (Plataforma de hardware do dispositivo), escolha a plataforma de hardware.

    > **Note**
    >
    > Apenas plataformas de hardware qualificadas para o FreeRTOS são exibidas nesta lista. Se estiver testando uma plataforma não qualificada e estiver usando o ciphersuite ECDSA P-256 SHA-256 para fazer login, você poderá escolher o perfil de assinatura de código do Simulador do Windows para produzir uma assinatura compatível. Se estiver usando uma plataforma não qualificada e estiver usando um ciphersuite diferente de ECDSA P-256 SHA-256 para assinar, você poderá usar o Code Signing para AWS IoT ou poderá assinar a atualização do firmware por conta própria. Para obter mais informações, consulte Assinatura digital da atualização de firmware.

    b. Em Certificado de assinatura de código, escolha Selecionar um certificado existente e, em seguida, selecione um certificado importado anteriormente ou escolha Importar um novo certificado de assinatura de código; escolha os arquivos e selecione Importar para importar um novo certificado.

    c. Em Pathname of code signing certificate on device (Nome do caminho do certificado de assinatura de código no dispositivo), insira o nome do caminho totalmente qualificado para o certificado de assinatura de código no dispositivo. Na maioria dos dispositivos, você pode

deixar esse campo em branco. No simulador do Windows e em dispositivos que colocam o certificado em um local de arquivo específico, insira o nome do caminho aqui.

> **Important**
>
> No Texas Instruments CC3220SF-LAUNCHXL, não inclua um caractere de barra (**/**) na frente do nome do arquivo se o certificado de assinatura de código existir na raiz do sistema de arquivos. Caso contrário, a atualização OTA falhará durante a autenticação com um erro `file not found`.

d. Escolha Create (Criar).

10. Em Arquivo, selecione Selecionar um arquivo existente e escolha Procurar S3. Uma lista dos buckets do Amazon S3 será exibida. Escolha o bucket que contém a atualização do firmware e escolha a atualização do firmware no bucket.

> **Note**
>
> Os projetos de demonstração do Microchip Curiosity PIC32MZEF produzem duas imagens binárias com nomes padrão de `mplab.production.bin` e `mplab.production.ota.bin`. Use o segundo arquivo ao fazer upload de uma imagem para a atualização OTA.

11. Em Nome do caminho do arquivo no dispositivo, insira o nome do caminho totalmente qualificado no local do dispositivo em que o trabalho OTA copiará a imagem do firmware. Esse local varia de acordo com a plataforma.

> **Important**
>
> No Texas Instruments CC3220SF-LAUNCHXL, devido a restrições de segurança, o nome do caminho da imagem do firmware deve ser `/sys/mcuflashimg.bin`.

12. Abra Tipo de arquivo e insira um valor inteiro no intervalo de 0 a 255. O tipo de arquivo inserido será adicionado ao documento do trabalho que é entregue ao MCU. Os desenvolvedores de firmware/software do MCU tem autonomia plena sobre o que fazer com esse valor. Os cenários possíveis incluem um MCU com um processador secundário que tem um firmware que pode ser atualizado independentemente do processador primário. Quando o dispositivo recebe um trabalho de atualização OTA, ele pode usar o Tipo de arquivo para identificar para qual processador a atualização se destina.

13. Em Perfil do IAM, escolha um perfil de acordo com as instruções em Criação de uma função de serviço de atualização de OTA.
14. Escolha Next (Próximo).
15. Insira um ID e uma descrição para seu trabalho de atualização OTA.
16. Em Job type (Tipo de trabalho), escolha Your job will complete after deploying to the selected devices/groups (snapshot) (Seu trabalho será concluído após a implantação nos dispositivos/grupos selecionados (snapshot)).
17. Escolha as configurações opcionais adequadas para o trabalho (Implementação de execuções de trabalho, Anulação do trabalho, Tempo limite das execuções dos trabalho e Tags).
18. Escolha Create (Criar).

Para usar uma imagem de firmware assinada anteriormente

1. Em Select and sign your firmware image (Selecionar e assinar a imagem de firmware), escolha Select a previously signed firmware image (Selecionar uma imagem de firmware previamente assinada).
2. Em Pathname of firmware image on device (Nome do caminho da imagem do firmware no dispositivo), insira o nome do caminho totalmente qualificado no local no dispositivo em que o trabalho OTA copiará a imagem do firmware. Esse local varia de acordo com a plataforma.
3. Em Previous code signing job (Trabalho de assinatura de código anterior), escolha Select (Selecionar) e, em seguida, escolha o trabalho de assinatura de código anterior usado para assinar a imagem de firmware que você está usando para a atualização OTA.

Usar a imagem de firmware assinada personalizada

1. Em Select and sign your firmware image (Selecionar e assinar a imagem de firmware), escolha Use my custom signed firmware image (Usar minha imagem de firmware assinada personalizada).
2. Em Pathname of code signing certificate on device (Nome do caminho do certificado de assinatura de código no dispositivo), insira o nome do caminho totalmente qualificado para o certificado de assinatura de código no dispositivo. Na maioria dos dispositivos, você pode deixar esse campo em branco. No simulador do Windows e em dispositivos que colocam o certificado em um local de arquivo específico, insira o nome do caminho aqui.

3. Em Pathname of firmware image on device (Nome do caminho da imagem do firmware no dispositivo), insira o nome do caminho totalmente qualificado no local no dispositivo em que o trabalho OTA copiará a imagem do firmware. Esse local varia de acordo com a plataforma.
4. Em Signature (Assinatura), cole sua assinatura no formato PEM.
5. Em Original hash algorithm (Algoritmo hash original), escolha o algoritmo hash que foi usado quando você criou sua assinatura de arquivo.
6. Em Original encryption algorithm (Algoritmo de criptografia original), escolha o algoritmo que foi usado quando você criou sua assinatura de arquivo.
7. Em Selecionar a imagem de firmware no S3, escolha o bucket do Amazon S3 e a imagem de firmware assinada no bucket do Amazon S3.

Depois de especificar as informações de assinatura de código, especifique o tipo de trabalho de atualização OTA, a função de serviço e um ID para sua atualização.

> **Note**
>
> Não use nenhuma informação pessoal identificável no ID de trabalho para a atualização OTA. Exemplos de informações de identificação pessoal incluem:
>
> - Nomes.
> - Endereços IP.
> - Endereços de e-mail.
> - Locais.
> - Dados bancários.
> - Informações médicas.

1. Em Job type (Tipo de trabalho), escolha Your job will complete after deploying to the selected devices/groups (snapshot) (Seu trabalho será concluído após a implantação nos dispositivos/grupos selecionados (snapshot)).
2. Em IAM role for OTA update job (Função do IAM para o trabalho de atualização OTA), escolha a função de serviço OTA.
3. Insira um ID alfanumérico para o seu trabalho e escolha Create (Criar).

O trabalho é exibido no console da AWS IoT com um status IN PROGRESS (EM PROGRESSO).

> **Note**
>
> - O console da AWS IoT não atualiza o estado dos trabalhos automaticamente. Atualize o navegador para ver as atualizações.

Conecte o terminal série UART ao seu dispositivo. Você verá a saída que indica que o dispositivo está fazendo download do firmware atualizado.

Depois que o dispositivo faz o download do firmware atualizado, ele reinicia e instala o firmware. Você pode ver o que está acontecendo no terminal UART.

Para obter um tutorial que mostre como usar o console para criar uma atualização OTA, consulte Aplicativo de demonstração de atualizações remotas.

Criação de uma atualização do OTA com a AWS CLI

Ao usar a AWS CLI para criar uma atualização OTA, você:

1. Assinar digitalmente a imagem de firmware.
2. Criar um fluxo da imagem de firmware assinada digitalmente.
3. Iniciar um trabalho de atualização OTA.

Assinatura digital da atualização de firmware

Ao usar a AWS CLI para executar atualizações OTA, você pode usar o Code Signing para AWS IoT ou pode assinar a atualização de firmware por conta própria. Para obter uma lista dos algoritmos de hashing e assinatura criptográfica compatíveis com o Code Signing para AWS IoT, consulte SigningConfigurationOverrides. Se desejar usar um algoritmo de criptografia que não é compatível com o Code Signing para AWS IoT, você precisará assinar o binário de firmware antes de fazer upload dele para o Amazon S3.

Assinatura de imagem de firmware com o Code Signing para AWS IoT

Para assinar a imagem de firmware usando o Code Signing para AWS IoT, você pode usar um dos SDKs ou ferramentas da linha de comando da AWS. Para obter mais informações sobre o Code Signing para AWS IoT, consulte Code Signing para AWS IoT.

Depois de instalar e configurar as ferramentas de assinatura de código, copie a imagem de firmware não assinada no bucket do Amazon S3 e inicie um trabalho de assinatura de código com os seguintes comandos da AWS CLI. O comando put-signing-profile cria um perfil de assinatura de código reutilizável. O comando start-signing-job inicia o trabalho de assinatura.

```
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**
>
> *your-source-bucket-name* e *your-destination-bucket-name* podem ser o mesmo bucket do Amazon S3.

Esses são os parâmetros para os comandos put-signing-profile e start-signing-job:

`source`

Especifica o local do firmware não assinado em um bucket do S3.

- `bucketName`: o nome do bucket do S3.
- `key`: a chave (nome do arquivo) do firmware no seu bucket do S3.
- `version`: a versão do S3 do firmware no seu bucket do S3. Isso é diferente da versão de firmware. Você pode encontrá-la navegando até o console do Amazon S3, escolhendo o bucket e, na parte superior da página, próximo a Versões, escolhendo Exibir.

`destination`

O destino no dispositivo para o qual o firmware assinado no bucket do S3 será copiado. O formato deste parâmetro é o mesmo do parâmetro `source`.

`signing-material`

O ARN do certificado de assinatura de código. Esse ARN é gerado quando você importa o certificado para o ACM.

`signing-parameters`

Um mapa de pares de chave/valor para assinatura. Podem incluir qualquer informação que você deseja usar durante a assinatura.

> **Note**
>
> Esse parâmetro é necessário ao criar um perfil de assinatura de código para assinar atualizações OTA com o Code Signing para AWS IoT.

`platform`

O `platformId` da plataforma de hardware para a qual você está distribuindo a atualização OTA.

Para retornar uma lista das plataformas disponíveis e seus valores de `platformId`, use o comando `aws signer list-signing-platforms`.

O trabalho de assinatura inicia e grava a imagem de firmware assinada no bucket de destino do Amazon S3. O nome do arquivo para a imagem de firmware assinada é um GUID. Você precisa desse nome de arquivo ao criar um fluxo. Você pode encontrar o nome do arquivo navegando até o console do Amazon S3 e escolhendo o bucket. Se você não vir um arquivo com um nome de arquivo GUID, atualize o navegador.

O comando exibe um ARN de trabalho e um ID de trabalho. Você precisará desses valores mais tarde. Para obter mais informações sobre o Code Signing para AWS IoT, consulte Code Signing para AWS IoT.

Assinatura de imagem de firmware manualmente

Assine digitalmente a imagem de firmware e faça upload da imagem de firmware assinada no bucket do Amazon S3.

Criação de um fluxo da atualização de firmware

Um fluxo é uma interface abstrata para dados que podem ser consumidos por um dispositivo. Um fluxo pode ocultar a complexidade de acessar dados armazenados em diferentes locais ou serviços

baseados em nuvem diferentes. O serviço Gerenciador de atualizações OTA permite que você use vários dados armazenados em vários locais no Amazon S3 para realizar uma atualização OTA.

Ao criar uma atualização OTA em AWS IoT, você também pode criar um fluxo que contém a atualização de firmware assinada. Crie um arquivo JSON (`stream.json`) que identifique a imagem de firmware assinada. O arquivo JSON deve conter o seguinte:

```
[
  {
    "fileId":"your_file_id",
    "s3Location":{
      "bucket":"your_bucket_name",
      "key":"your_s3_object_key"
    }
  }
]
```

Estes são os atributos no arquivo JSON:

`fileId`

Um inteiro arbitrário entre 0 a 255 que identifica a imagem de firmware.

`s3Location`

O bucket e a chave para o firmware ser transmitido.

`bucket`

O bucket do Amazon S3 no qual está armazenada a imagem de firmware não assinada.

`key`

O nome do arquivo da imagem de firmware assinada no bucket do Amazon S3. Você pode encontrar esse valor no console do Amazon S3, observando o conteúdo do bucket.

Se você estiver usando o Code Signing para AWS IoT, o nome do arquivo será um GUID gerado pelo Code Signing para AWS IoT.

Use o comando create-stream da AWS CLI para criar um fluxo.

```
aws iot create-stream \
    --stream-id your_stream_id \
```

```
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

Estes são os argumentos para o comando create-stream da AWS CLI:

`stream-id`

Uma string arbitrária para identificar o fluxo.

`description`

Uma descrição opcional do fluxo.

`files`

Uma ou mais referências a arquivos JSON que contêm dados sobre imagens de firmware a serem transmitidas. O arquivo JSON deve conter os atributos a seguir:

`fileId`

Um ID de arquivo arbitrário.

`s3Location`

O nome do bucket onde a imagem de firmware assinada está armazenada e a chave (nome do arquivo) da imagem de firmware assinada.

`bucket`

O bucket do Amazon S3 onde a imagem de firmware assinada está armazenada.

`key`

A chave (nome do arquivo) da imagem de firmware assinada.

Quando você usa o Code Signing para AWS IoT, essa chave é um GUID.

Veja a seguir um exemplo de arquivo `stream.json`.

```
[
    {
        "fileId":123,
        "s3Location": {
            "bucket":"codesign-ota-bucket",
            "key":"48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
```

```
        }
    }
]
```

`role-arn`

O perfil de serviço OTA que também concede acesso ao bucket do Amazon S3 em que a imagem do firmware está armazenada.

Para encontrar a chave de objeto do Amazon S3 da imagem de firmware assinada, use o comando aws signer describe-signing-job --job-id ***`my-job-id`***, em que `my-job-id` é o ID do trabalho exibido pelo comando create-signing-job da AWS CLI. A saída do comando describe-signing-job contém a chave da imagem de firmware assinada.

```
... text deleted for brevity ...
  "signedObject": {
    "s3": {
      "bucketName": "ota-bucket",
      "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
  }
... text deleted for brevity ...
```

Criação de uma atualização do OTA

Use o comando create-ota-update da AWS CLI para criar um trabalho de atualização OTA.

> **Note**
>
> Não use nenhuma informação pessoal identificável (PII) no ID de trabalho de atualização OTA. Exemplos de informações de identificação pessoal incluem:
>
> - Nomes.
> - Endereços IP.
> - Endereços de e-mail.
> - Locais.
> - Dados bancários.
> - Informações médicas.

```
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 de `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 de **`cli-input-json`**

Nome	Type	Descrição
`otaUpdateId`	string (máx: 128 mín:1)	O ID da atualização OTA a ser criado.
`description`	string (máx: 2028)	A descrição da atualização OTA.

Nome	Type	Descrição
`targets`	list	Os dispositivos de destino para receber atualizações OTA.
`protocols`	list	O protocolo usado para transferir a imagem de atualização OTA. Os valores válidos são [HTTP], [MQTT], [HTTP, MQTT]. Quando HTTP e MQTT são especificados, o dispositivo de destino pode escolher o protocolo.

Nome	Type	Descrição
`targetSelection`	string	Especifica se a atualização continuará a ser executada (CONTINUOUS) ou se será concluída depois que todas as coisas especificadas como destinos tiverem concluído a atualização (SNAPSHOT). Se contínua, a atualização também poderá ser executada em uma coisa quando uma alteração for detectada em um destino. Por exemplo, uma atualização será executada em uma coisa quando a coisa for adicionada a um grupo de destinos, mesmo depois da atualização ter sido concluída por todas as coisas originalmente no grupo. Valores válidos: CONTINUOUS | SNAPSHOT. enum: CONTINUOUS | SNAPSHOT
`awsJobExecutionsRolloutConfig`		Configuração da distribuição de atualizações OTA.
`maximumPerMinute`	integer (máx: 1000 mín:1)	O número máximo de execuções de trabalhos de atualizações OTA iniciadas por minuto.

Nome	Type	Descrição
`exponentialRate`		A taxa de aumento para a implantação de um trabalho. Esse parâmetro permite definir um aumento na taxa exponencial para a implantação de um trabalho.
`baseRatePerMinute`	integer (máx: 1000 mín:1)	O número mínimo de coisas que serão notificadas de um serviço pendente, por minuto, no início da implantação do trabalho. Esta é a taxa inicial da implantação.
`rateIncreaseCriteria`		Os critérios para iniciar o aumento na taxa de implantação de um trabalho. O AWS IoT oferece suporte a até um dígito após o ponto decimal (por exemplo, 1,5, mas não 1,55).
`numberOfNotifiedThings`	integer (mín:1)	Quando esse número de coisas tiver sido notificado, ele iniciará um aumento na taxa de implantação.
`numberOfSucceededThings`	integer (mín:1)	Quando esse número de coisas tiver sido bem-sucedido na execução de trabalho, ele iniciará um aumento na taxa de implantação.
`awsJobPresignedUrlConfig`		Informações da configuração de pre-signed URLs.

Nome	Type	Descrição
`expiresInSec`	longo	Por quanto tempo (em segundos) as pre-signed URLs são válidas. Os valores válidos são 60 – 3600, o valor padrão é 1800 segundos. Os pre-signed URLs são gerados quando uma solicitação para o documento de trabalho é recebida.
`awsJobAbortConfig`		Os critérios que determina m quando e como ocorre a interrupção de um trabalho.
`abortCriteriaList`	list	A lista de critérios que determinam quando e como interromper o trabalho.
`failureType`	string	O tipo de falhas de execução de trabalho que podem iniciar uma interrupção. enum: FAILED | REJECTED | TIMED_OUT | ALL
`action`	string	O tipo de ação do trabalho a ser executada para iniciar a interrupção. enum: CANCEL
`minNumberOfExecutedThings`	integer (mín:1)	O número mínimo de coisas que devem receber notificaç ões de execução de trabalho antes que ele possa ser interrompido.

Nome	Type	Descrição
`awsJobTimeoutConfig`		Especifica o tempo que cada dispositivo tem para concluir a execução do trabalho. Um temporizador é iniciado quando o status da execução do trabalho é definido como `IN_PROGRESS` . Se o status da execução do trabalho não estiver definido como outro estado final antes que o temporizador expire, ele será definido automaticamente como `TIMED_OUT` .
`inProgressTimeoutInMinutes`	longo	Especifica o tempo, em minutos, que este dispositivo tem para concluir a execução do trabalho. O intervalo de tempo limite pode estar em qualquer lugar entre 1 minuto e 7 dias (1 a 10.080 minutos). O temporizador em andamento não pode ser atualizado e será aplicado a todas as execuções do trabalho. Sempre que uma execução de trabalho permanecer com o status IN_PROGRESS por mais tempo que esse intervalo, a execução falhará e alternará para o status `TIMED_OUT` final.

Nome	Type	Descrição
`files`	list	Os arquivos a serem transmitidos pela atualização OTA.
`fileName`	string	O nome do arquivo.
`fileType`	integer range- máx.: 255, mín.: 0	Um valor inteiro que você pode incluir no documento de trabalho para permitir que os dispositivos identifiquem o tipo de arquivo recebido da nuvem.
`fileVersion`	string	A versão do arquivo.
`fileLocation`		O local do firmware atualizado.
`stream`		O stream que contém a atualização OTA.
`streamId`	string (máx: 128 mín:1)	O ID do fluxo.
`fileId`	integer (máx: 255 mín:0)	O ID de um arquivo associado a um fluxo.
`s3Location`		O local do firmware atualizado no S3.
`bucket`	string (mín:1)	O bucket do S3.
`key`	string (mín:1)	A chave de S3.

Nome	Type	Descrição
`version`	string	A versão do bucket do S3.
`codeSigning`		O método de assinatura do código do arquivo.
`awsSignerJobId`	string	O ID da AWSSignerJob que foi criado para assinar o arquivo.
`startSigningJobParameter`		Descreve o trabalho de assinatura de código.
`signingProfileParameter`		Descreve o perfil de assinatura de código.
`certificateArn`	string	ARN de certificado.
`platform`	string	A plataforma de hardware do dispositivo.
`certificatePathOnDevice`	string	A localização do certificado de assinatura de código no dispositivo.
`signingProfileName`	string	O nome do perfil de assinatura de código.
`destination`		O local para gravar o arquivo de código assinado.
`s3Destination`		Descreve o local no S3 do firmware atualizado.
`bucket`	string (mín:1)	O bucket do S3 que contém o firmware atualizado.
`prefix`	string	O prefixo do S3.

Nome	Type	Descrição
`customCodeSigning`		Um método personalizado para assinar um arquivo por código.
`signature`		A assinatura do arquivo.
`inlineDocument`	blob	A representação binária codificada em base64 da assinatura por código.
`certificateChain`		A cadeia de certificados.
`certificateName`	string	O nome do certificado.
`inlineDocument`	string	A representação binária codificada em base64 da cadeia de certificados de assinaturas por código.
`hashAlgorithm`	string	O algoritmo de hash usado para assinar o arquivo por código.
`signatureAlgorithm`	string	O algoritmo de assinatura usado para assinar o arquivo por código.
`attributes`	mapear	Uma lista de pares nome/atri buto.
`roleArn`	string (máx: 2048 mín:20)	O perfil do IAM que concede acesso ao AWS IoT para o Amazon S3, os trabalhos do AWS IoT e os recursos do AWS Code Signing para criar um trabalho de atualização OTA.

Nome	Type	Descrição
`additionalParameters`	mapear	Uma lista de parâmetros de atualização OTA adicionais que são pares nome-valor.
`tags`	list	Metadados que podem ser usados para gerenciar atualizações.
`Key`	string (máx: 128 mín:1)	A chave da tag.
`Value`	string (máx: 256 mín:1)	O valor da tag.

Resultado

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

Campos de saída da AWS CLI

Nome	Type	Descrição
`otaUpdateId`	string (máx: 128 mín:1)	O ID da atualização OTA
`awsIotJobId`	string	O ID do trabalho do AWS IoT associado à atualização OTA.
`otaUpdateArn`	string	O ARN da atualização OTA.

Nome	Type	Descrição
`awsIotJobArn`	string	O ARN do trabalho do AWS IoT associado à atualização OTA.
`otaUpdateStatus`	string	O status da atualização OTA. enum: CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

Veja a seguir um exemplo de um arquivo JSON transmitido para o comando create-ota-update que utiliza o Code Signing para AWS IoT.

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

Veja a seguir um exemplo de um arquivo JSON transmitido para o comando create-ota-update da AWS CLI que usa um arquivo em linha para fornecer material de assinatura de código personalizado.

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

Veja a seguir um exemplo de um arquivo JSON transmitido para o comando create-ota-update da AWS CLI que permite que o FreeRTOS OTA inicie um trabalho de assinatura de código e crie um perfil de assinatura de código e fluxo.

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

Veja a seguir um exemplo de um arquivo JSON transmitido para o comando create-ota-update da AWS CLI que cria uma atualização OTA que inicia um trabalho de assinatura de código com um perfil existente e usa o fluxo especificado.

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

Veja a seguir um exemplo de um arquivo JSON transmitido para o comando create-ota-update da AWS CLI que permite que o FreeRTOS OTA crie um fluxo com um ID de trabalho de assinatura de código existente.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

Veja a seguir um exemplo de um arquivo JSON transmitido para o comando create-ota-update da AWS CLI que cria uma atualização OTA. A atualização cria um fluxo a partir do objeto especificado do S3 e usa a assinatura de código personalizado.

```
[
  {
    "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"
```

```
      }
    }
  }
]
```

Listagem de atualizações do OTA

Você pode usar o comando list-ota-updates da AWS CLI para obter uma lista de todas as atualizações OTA.

```
aws iot list-ota-updates
```

A saída do comando list-ota-updates parece com o exemplo a seguir.

```
{
  "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
    }
  ]
}
```

Obtenção de informações sobre uma atualização OTA

Você pode usar o comando get-ota-update da AWS CLI para obter o status da criação ou exclusão de uma atualização OTA.

```
aws iot get-ota-update --ota-update-id your-ota-update-id
```

A saída do comando get-ota-update é semelhante ao exemplo a seguir.

```
{
    "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"
    }
}
```

Os valores retornados para `otaUpdateStatus` incluem o seguinte:

`CREATE_PENDING`

A criação de uma atualização OTA está pendente.

`CREATE_IN_PROGRESS`

Uma atualização OTA está sendo criada.

`CREATE_COMPLETE`

Uma atualização OTA foi criada.

`CREATE_FAILED`

A criação de uma atualização OTA falhou.

`DELETE_IN_PROGRESS`

Uma atualização OTA está sendo excluída.

`DELETE_FAILED`

A exclusão de uma atualização OTA falhou.

> **Note**
>
> Para obter o status de execução de uma atualização OTA após sua criação, você precisará usar o comando describe-job-execution. Para obter mais informações, consulte Descrever a execução do trabalho.

Exclusão de dados relacionados ao OTA

No momento, não é possível usar o console da AWS IoT para excluir fluxos ou atualizações OTA. Você pode usar a AWS CLI para excluir fluxos, atualizações OTA e trabalhos da AWS IoT criados durante uma atualização OTA.

Exclusão de um fluxo do OTA

Quando você cria uma atualização OTA que usa MQTT, você pode usar a linha de comando ou o console da AWS IoT para criar um fluxo para dividir o firmware em pedaços para que ele possa ser enviado pelo MQTT. Você pode excluir esse fluxo com o comando delete-stream da AWS CLI, conforme mostrado no exemplo a seguir.

```
aws iot delete-stream --stream-id your_stream_id
```

Como excluir uma atualização do OTA

Quando você cria uma atualização OTA, são criados:

- Uma entrada no banco de dados de trabalhos de atualização OTA.
- Um trabalho da AWS IoT para executar a atualização.
- Uma execução de trabalho da AWS IoT para cada dispositivo que está sendo atualizado.

O comando delete-ota-update exclui a entrada somente no banco de dados de trabalhos de atualização OTA. É necessário usar o comando delete-job para excluir o trabalho da AWS IoT.

Use o comando delete-ota-update para excluir uma atualização OTA.

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

`ota-update-id`

O ID da atualização OTA a ser excluído.

`delete-stream`

Exclui o fluxo associado à atualização OTA.

`force-delete-aws-job`

Exclui o trabalho da AWS IoT associado à atualização OTA. Se esse sinalizador não estiver definido e o trabalho estiver no estado `In_Progress`, o trabalho não será excluído.

Exclusão de um trabalho da IoT criado para uma atualização do OTA

O FreeRTOS cria um trabalho do AWS IoT quando você cria uma atualização OTA. Uma execução de trabalho também é criada para cada dispositivo que processa o trabalho. Você pode usar o comando delete-job da AWS CLI para excluir um trabalho e as execuções de trabalho associadas.

```
aws iot delete-job --job-id your-job-id --no-force
```

O parâmetro `no-force` especifica que somente os trabalhos que estão em estado terminal (COMPLETED ou CANCELED) podem ser excluídos. Você pode excluir um trabalho que está em um estado não terminal, transmitindo o parâmetro `force`. Para obter mais informações, consulte API DeleteJob.

> **Note**
>
> A exclusão de um trabalho com o status IN_PROGRESS interrompe todas as execuções de trabalho que estão em IN_PROGRESS nos dispositivos e pode resultar na permanência de um dispositivo em um estado não determinista. Certifique-se de que cada dispositivo que esteja executando um trabalho que foi excluído possa retornar a um estado conhecido.

Dependendo do número de execuções de trabalho criadas para o trabalho e de outros fatores, pode levar alguns minutos para excluir um trabalho. Enquanto o trabalho estiver sendo excluído, seu status será DELETION_IN_PROGRESS. A tentativa de excluir ou cancelar um trabalho cujo status já seja DELETION_IN_PROGRESS resulta em um erro.

Você pode usar a delete-job-execution para excluir uma execução de trabalho. Você pode querer excluir uma execução de trabalho quando um pequeno número de dispositivos não puder processar um trabalho. Isso exclui a execução do trabalho de um único dispositivo, conforme mostrado no exemplo a seguir.

```
aws iot delete-job-execution --job-id your-job-id --thing-name
                    your-thing-name --execution-number your-job-execution-number --no-
force
```

Assim como ocorre com o comando delete-job da AWS CLI, você pode transmitir o parâmetro `--force` para o delete-job-execution forçar a exclusão de um trabalho de execução. Para obter mais informações, consulte API DeleteJobExecution.

> **Note**
>
> A exclusão de uma execução de trabalho com o status IN_PROGRESS interrompe todas as execuções de trabalho que estão em IN_PROGRESS nos dispositivos e pode resultar na permanência de um dispositivo em um estado não determinista. Certifique-se de que cada dispositivo que esteja executando um trabalho que foi excluído possa retornar a um estado conhecido.

Para obter mais informações sobre como usar o aplicativo de demonstração de atualização OTA, consulte Aplicativo de demonstração de atualizações remotas.

Serviço do gerenciador de atualização OTA

O serviço do gerenciador de atualização over-the-air (OTA) fornece uma maneira de:

- Criar uma atualização OTA e os recursos usados, incluindo um trabalho de AWS IoT, um fluxo AWS IoT e a assinatura de código.
- Obtenha informações sobre uma atualização OTA.
- Liste todas as atualizações OTA associadas à sua conta da AWS.
- Excluir uma atualização OTA.

Uma atualização OTA é uma estrutura de dados mantida pelo serviço do gerenciador de atualização OTA. Ela contém:

- Um ID de atualização OTA.
- Uma descrição opcional da atualização OTA.
- Uma lista de dispositivos a serem atualizados (destinos)
- O tipo de atualização OTA: CONTINUOUS ou SNAPSHOT. Consulte a seção Trabalhos do Guia do desenvolvedor do AWS IoT para obter uma discussão sobre o tipo de atualização que você precisa.
- O protocolo usado para executar a atualização OTA: [MQTT], [HTTP] ou [MQTT, HTTP]. Quando você especifica MQTT e HTTP, a configuração do dispositivo determina o protocolo usado.
- Uma lista de arquivos a serem enviados aos dispositivos de destino.
- O perfil do IAM que concede acesso ao AWS IoT para o Amazon S3, os trabalhos do AWS IoT e os recursos do AWS Code Signing para criar um trabalho de atualização OTA.

- Uma lista opcional de pares de nome/valor definida pelo usuário.

As atualizações OTA foram projetadas para atualizar o firmware do dispositivo, mas você pode usá-las para enviar qualquer arquivo que você queira para um ou mais dispositivos registrados na AWS IoT. Quando você envia atualizações de firmware over-the-air, recomendamos que você as assine digitalmente para que os dispositivos que as recebem possam verificar se não foram adulteradas no caminho.

Você pode enviar imagens de firmware atualizadas usando o protocolo HTTP ou MQTT, dependendo das configurações que você escolher. Você pode assinar suas atualizações de firmware com o Code Signing para FreeRTOS ou usar suas próprias ferramentas de assinatura de código.

Para obter mais controle sobre o processo, você pode usar a API CreateStream para criar um fluxo ao enviar atualizações pelo MQTT. Em alguns casos, você pode modificar o código do agente do FreeRTOS para ajustar o tamanho dos blocos que você envia e recebe.

Ao criar uma atualização OTA, o serviço gerenciador OTA cria um trabalho da AWS IoT para notificar os dispositivos de que uma atualização está disponível. O agente OTA do FreeRTOS é executado em seus dispositivos e detecta mensagens de atualização. Quando uma atualização está disponível, ele solicita a imagem de atualização de firmware por HTTP ou MQTT e armazena os arquivos localmente. Ele verifica a assinatura digital dos arquivos obtidos por download e, se for válida, instala a atualização do firmware. Se não estiver usando o FreeRTOS, será necessário implementar seu próprio agente OTA para detectar e fazer download das atualizações e realizar operações de instalação.

Integração do agente OTA ao aplicativo

O agente remoto (OTA) foi projetado para simplificar a quantidade de código que você precisa gravar para adicionar a funcionalidade de atualização OTA ao produto. Essa carga de integração consiste principalmente na inicialização do agente OTA e, opcionalmente, na criação de uma função de retorno de chamada personalizada para responder às mensagens do evento de conclusão OTA. Durante a inicialização, as interfaces do sistema operacional, MQTT, HTTP (se o HTTP for usado para download de arquivos) e de implementação específica da plataforma (PAL) são passadas para o agente OTA. Os buffers também podem ser inicializados e passados para o agente OTA.

> **Note**
>
> Embora a integração do recurso de atualização OTA no aplicativo seja bastante simples, o sistema de atualização OTA exige uma compreensão mais ampla do que apenas integração

de código de dispositivo. Para saber como configurar sua conta da AWS com as coisas da AWS IoT, as credenciais, os certificados de assinatura de código, os dispositivos de provisionamento e os trabalhos de atualização OTA, consulte Pré-requisitos do FreeRTOS.

Gerenciamento de conexões

O agente OTA usa o protocolo MQTT para todas as operações de comunicação de controle envolvendo serviços de AWS IoT, mas não gerencia a conexão MQTT. Para assegurar que o agente OTA não interfira na política de gerenciamento de conexão do aplicativo, a conexão MQTT (incluindo a desconexão e qualquer funcionalidade de reconexão) deve ser processada pelo aplicativo do usuário principal. O arquivo pode ser obtido por download através do protocolo MQTT ou HTTP. Você pode escolher o protocolo ao criar o trabalho OTA. Se você escolher MQTT, o agente OTA usará a mesma conexão para operações de controle e para fazer download de arquivos.

Demonstração OTA simples

O exemplo a seguir é um trecho de uma demonstração de OTA simples que mostra como o agente se conecta ao agente MQTT e inicializa o agente OTA. Nesse exemplo, configuramos a demonstração para usar o retorno de chamada de aplicativo OTA padrão e para retornar algumas estatísticas uma vez por segundo. Para não estender a explicação, deixamos de fora alguns detalhes desta demonstração.

A demonstração OTA também demonstra o gerenciamento da conexão MQTT monitorando o retorno de chamada de desconexão e restabelecendo a conexão. Quando ocorre uma desconexão, a demonstração primeiro suspende as operações do agente OTA e depois tenta restabelecer a conexão MQTT. As tentativas de reconexão do MQTT são atrasadas por um tempo que aumenta exponencialmente até um valor máximo e uma tremulação também é adicionada. Se a conexão for restabelecida, o agente OTA continuará suas operações.

Para um exemplo de trabalho que usa o agente MQTT da AWS IoT, consulte o código de demonstração OTA no diretório `demos/ota`.

Como o agente OTA é sua própria tarefa, o atraso intencional de um segundo nesse exemplo afeta apenas esse aplicativo. Não tem impacto no desempenho do 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;
}
```

Veja a seguir o fluxo de alto nível deste aplicativo de demonstração:

- Crie um contexto do agente MQTT.
- Conecte-se ao endpoint da AWS IoT.
- Inicialize o agente OTA.
- Loop que permite um trabalho de atualização OTA e estatísticas de saídas uma vez por segundo.
- Se o MQTT se desconectar, suspenda as operações do agente OTA.
- Tente se conectar novamente com atraso e tremulação exponencial.
- Se reconectado, retome as operações do agente OTA.
- Se o agente parar, espere um segundo e tente se reconectar.

Uso do retorno de chamada do aplicativo para eventos do agente OTA

O exemplo anterior usou `prvOtaAppCallback` como manipulador de retorno de chamada para eventos do agente OTA. (Veja o quarto parâmetro da chamada de API `OTA_Init`). Se desejar implementar o tratamento personalizado dos eventos de conclusão, deverá alterar o tratamento padrão na demonstração/aplicativo OTA. Durante o processo do OTA, o agente OTA pode enviar um dos seguintes enums de eventos ao manipulador de retorno de chamada. Cabe ao desenvolvedor do aplicativo decidir como e quando processar esses 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;
```

O agente OTA pode receber uma atualização em segundo plano durante o processamento ativo do aplicativo principal. A finalidade de fornecer esses eventos é permitir que o aplicativo decida se uma ação pode ser executada imediatamente ou se deve ser adiada até após a conclusão de algum outro processamento específico do aplicativo. Isso evita uma interrupção imprevista do dispositivo durante o processamento ativo (por exemplo, vacuum) que seria causado por uma redefinição após uma atualização de firmware. Estes são os eventos de trabalho recebidos pelo manipulador de retorno de chamada:

`OtaJobEventActivate`

Quando o manipulador de retorno de chamada recebe esse evento, você pode redefinir o dispositivo imediatamente ou agendar uma chamada para redefinir o dispositivo posteriormente. Isso permite que você adie a redefinição do dispositivo e a fase de teste automático, se necessário.

`OtaJobEventFail`

A atualização falhou quando o manipulador de retorno de chamada recebeu esse evento. Não é necessário fazer nada nesse caso. Você pode querer produzir uma mensagem de log ou fazer algo específico ao aplicativo.

`OtaJobEventStartTest`

A fase de teste automático destina-se a permitir que o firmware recém-atualizado execute e teste ele mesmo antes de determinar que está funcionando corretamente e confirmar que ele é a última imagem permanente do aplicativo. Quando uma nova atualização for recebida e autenticada e

o dispositivo tiver sido redefinido, o agente OTA enviará o evento `OtaJobEventStartTest` para a função de retorno de chamada quando estiver pronto para o teste. O desenvolvedor pode adicionar quaisquer testes necessários para determinar se o firmware do dispositivo está funcionando corretamente após a atualização. Quando o firmware do dispositivo é considerado confiável pelos testes automáticos, o código deve confirmar o firmware como a nova imagem permanente chamando a função `OTA_SetImageState( OtaImageStateAccepted )`.

`OtaJobEventProcessed`

O evento OTA enfileirado por `OTA_SignalEvent` é processado, portanto, operações de limpeza, como liberar os buffers OTA, podem ser realizadas.

`OtaJobEventSelfTestFailed`

O teste automático OTA falhou no trabalho atual. O tratamento padrão para esse evento é desligar o agente OTA e reiniciá-lo para que o dispositivo volte à imagem anterior.

`OtaJobEventUpdateComplete`

O evento de notificação para a conclusão da atualização do trabalho OTA.

Segurança de OTA

A seguir estão três aspectos da segurança over-the-air (OTA):

Segurança de conexão

O serviço OTA Update Manager depende dos mecanismos de segurança existentes, como a autenticação mútua do Transport Layer Security (TLS), usada pela AWS IoT. O tráfego de atualização OTA passa pelo gateway de dispositivo da AWS IoT e usa os mecanismos de segurança da AWS IoT. Cada mensagem HTTP ou MQTT recebida e enviada por meio do gateway de dispositivo é submetida a autenticação e autorização rigorosas.

Autenticidade e integridade das atualizações OTA

O firmware pode ser assinado digitalmente antes de uma atualização OTA para garantir que ele seja de uma fonte confiável e não tenha sido adulterado.

O serviço de gerenciador de atualização OTA do FreeRTOS usa o Code Signing do AWS IoT para assinar o firmware automaticamente. Para obter mais informações, consulte Code Signing para AWS IoT.

O agente OTA que é executado nos dispositivos executa verificações de integridade no firmware quando ele chega no dispositivo.

Segurança do operador

Cada chamada de API feita por meio da API do ambiente de gerenciamento é submetida à autenticação e autorização padrão do Signature Version 4 do IAM. Para criar uma implantação, é necessário ter permissões para invocar as APIs`CreateDeployment`, `CreateJob` e `CreateStream`. Além disso, na política de bucket do Amazon S3 ou na ACL, é necessário conceder permissões de leitura à entidade principal de serviço do AWS IoT para que a atualização do firmware armazenada no Amazon S3 possa ser acessada durante o streaming.

Code Signing para AWS IoT

O console do AWS IoT usa Code Signing para AWS IoT para assinar automaticamente a imagem de firmware para qualquer dispositivo compatível com o AWS IoT.

O Code Signing para AWS IoT usa um certificado e uma chave privada que você importa para o ACM. Você pode usar um certificado autoassinado para testes, mas recomendamos obter um certificado de uma autoridade de certificação (CA) comercial reconhecida.

Os certificados de assinatura de código usam as extensões `Key Usage` e `Extended Key Usage` do X.509 versão 3. A extensão `Key Usage` é definida como `Digital Signature` e a extensão `Extended Key Usage` é definida como `Code Signing`. Para obter mais informações sobre como assinar a imagem de código, consulte Guia do desenvolvedor do Code Signing para AWS IoT e Referência da API do Code Signing para AWS IoT.

> **Note**
>
> Você pode fazer download do Code Signing para AWS IoT SDK em Ferramentas para a Amazon Web Services.

Solução de problemas do OTA

As seções a seguir contêm informações para ajudar você a solucionar problemas com atualizações OTA.

Tópicos

- Configurar o CloudWatch Logs para atualizações OTA
- Registrar em log chamadas de API do OTA da AWS IoT com o AWS CloudTrail
- Obtenção de detalhes da falha CreateOTAUpdate usando a AWS CLI
- Obtenção de códigos de falha de OTA com a AWS CLI
- Solução de problemas de atualizações de OTA de vários dispositivos
- Solução de problemas de atualizações de OTA com o Launchpad CC3220SF da Texas Instruments

Configurar o CloudWatch Logs para atualizações OTA

O serviço de atualização OTA oferece suporte ao registro em log com o Amazon CloudWatch. Você pode usar o console do AWS IoT para habilitar e configurar o registro em log do Amazon CloudWatch para atualizações OTA. Para obter mais informações, consulte Cloudwatch Logs.

Para habilitar o registro em log, é necessário criar um perfil do IAM e configurar o registro em log de atualizações OTA.

> **Note**
>
> Antes de habilitar o registro em log de atualizações OTA, entenda as permissões de acesso do CloudWatch Logs. Os usuários com acesso ao CloudWatch Logs podem ver suas informações de depuração. Para obter informações, consulte Autenticação e controle de acesso para o Amazon CloudWatch Logs.

Criação de uma função de registro em log e habilitar o registro em log

Use o console do AWS IoT para criar uma função de registro em log e habilitar esse registro.

1. No painel de navegação, escolha Settings (Configurações).
2. Em Logs, escolha Edit (Editar).
3. Em Level of verbosity (Nível de verbosidade), escolha Debug (Depuração).
4. Em Definir perfil, escolha Criar para criar um perfil do IAM para o registro em log.
5. Em Name (Nome), insira um nome único para a função. A função será criada com todas as permissões necessárias.
6. Escolha Atualizar.

Logs de atualização de OTA

O serviço de atualização OTA publica logs na conta quando ocorre uma das situações a seguir:

- Uma atualização OTA é criada.
- Uma atualização OTA é concluída.
- Um trabalho de assinatura de código é criado.
- Um trabalho de assinatura de código é concluído.
- Um trabalho do AWS IoT é criado.
- Um trabalho do AWS IoT é concluído.
- Um fluxo é criado.

Você pode visualizar os logs no console do CloudWatch.

Como visualizar uma atualização OTA no CloudWatch Logs

1. Na página de navegação, escolha Logs.
2. Em Grupos de logs, escolha AWSIoTLogsV2.

Os logs de atualização OTA podem conter as seguintes propriedades:

accountId

O ID da conta da AWS no qual o log foi gerado.

actionType

A ação que gerou o log. Isso pode ser definido como um dos seguintes valores:

- `CreateOTAUpdate`: uma atualização OTA foi criada.
- `DeleteOTAUpdate`: uma atualização OTA foi excluída.
- `StartCodeSigning`: um trabalho de assinatura de código foi iniciado.
- `CreateAWSJob`: um trabalho da AWS IoT foi criado.
- `CreateStream`: um fluxo foi criado.
- `GetStream`: uma solicitação de stream foi enviada para o atributo de entrega de arquivos baseado em MQTT do AWS IoT.

- `DescribeStream`: uma solicitação de informações sobre um stream foi enviada para o atributo de entrega de arquivos baseado em MQTT do AWS IoT.

awsJobId

O ID de trabalho da AWS IoT que gerou o log.

clientId

O ID do cliente MQTT que fez a solicitação que gerou o log.

clientToken

O token do cliente associado à solicitação que gerou o log.

detalhes

Mais informações sobre a operação que gerou o log.

logLevel

O nível de registro do log. Para logs de atualização OTA, isso é sempre definido como `DEBUG`.

otaUpdateId

O ID da atualização OTA que gerou o log.

protocol

O protocolo usado para fazer a solicitação que gerou o log.

status

O status da operação que gerou o log. Os valores válidos são:

- Bem-sucedida
- Falha

streamId

O ID do fluxo da AWS IoT que gerou o log.

timestamp

A hora em que o log foi gerado.

topicName

Um tópico MQTT usado para fazer a solicitação que gerou o log.

Logs de exemplo

Veja a seguir um exemplo de log gerado quando um trabalho de assinatura de código é iniciado:

```
{
    "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."
}
```

Veja a seguir um exemplo de log gerado quando um trabalho da AWS IoT é criado:

```
{
    "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."
}
```

Veja a seguir um log de exemplo gerado quando uma atualização OTA é criada:

```
{
    "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."
}
```

Veja a seguir um log de exemplo gerado quando um fluxo é criado:

```
{
```

```
    "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."
}
```

Veja a seguir um log de exemplo gerado quando uma atualização OTA é excluída:

```
{
    "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."
}
```

Veja a seguir um log de exemplo gerado quando um dispositivo solicita um stream do atributo de entrega de arquivos baseado em 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."
}
```

Veja a seguir um exemplo de log gerado quando um dispositivo chama a 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 em log chamadas de API do OTA da AWS IoT com o AWS CloudTrail

O FreeRTOS é integrado ao CloudTrail, um serviço que captura chamadas de API do AWS IoT específicas e fornece arquivos de log a um bucket do Amazon S3 especificado por você. O CloudTrail captura chamadas de API do seu código para as APIs OTA do AWS IoT. Usando as informações coletadas pelo CloudTrail, você pode determinar a solicitação que foi feita para o OTA do AWS IoT, o endereço IP de origem a partir do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e assim por diante.

Para obter mais informações sobre o CloudTrail, incluindo como configurá-lo e habilitá-lo, consulte o Guia do usuário do AWS CloudTrail.

Informações dos planos do FreeRTOS no CloudTrail

Quando o registro em log do CloudTrail está habilitado em sua conta AWS, as chamadas de API feitas para ações do OTA do AWS IoT serão rastreadas nos arquivos de log do CloudTrail, onde serão gravadas com outros registros de serviço da AWS. O CloudTrail determina quando criar e gravar em um novo arquivo de acordo com o período e o tamanho do arquivo.

As seguintes ações de ambiente de gerenciamento OTA do AWS IoT são registradas pelo CloudTrail:

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream

- DeleteStream
- CreateOTAUpdate
- GetOTAUpdate
- ListOTAUpdates
- DeleteOTAUpdate

> **Note**
>
> As ações do plano de dados OTA do AWS IoT (lado do dispositivo) não são registradas em log pelo CloudTrail. Use o CloudWatch para monitorá-los.

Cada entrada de log contém informações sobre quem gerou a solicitação. As informações de identidade do usuário na entrada de log ajudam você a determinar o seguinte:

- Se a solicitação foi feita com credenciais de usuário raiz ou do usuário do IAM.
- Se a solicitação foi feita com credenciais de segurança temporárias de uma função ou de um usuário federado.
- Se a solicitação foi feita por outro serviço da AWS.

Para obter mais informações, consulte o Elemento userIdentity do CloudTrail. AWS IoT As ações OTA são documentadas na Referência da API OTA do AWS IoT.

Você pode armazenar os arquivos de log no seu bucket do Amazon S3 pelo tempo que desejar, mas também pode definir regras do ciclo de vida do Amazon S3 para arquivar ou excluir os arquivos de log automaticamente. Por padrão, os arquivos de log são criptografados com a criptografia do lado do servidor (SSE) do Amazon S3.

Se desejar ser notificado sobre quando os arquivos de log são entregues, é possível configurar o CloudTrail para publicar notificações do Amazon SNS. Para obter mais informações, consulte Configuração de notificações do Amazon SNS para o CloudTrail.

Também é possível agregar arquivos de log do OTA do AWS IoT de várias regiões da AWS e contas da AWS em um único bucket do Amazon S3.

Para obter mais informações, consulte Recebimento de arquivos de log do CloudTrail de várias regiões e Recebimento de arquivos de log do CloudTrail de várias contas.

Noções básicas das entradas de arquivo de log do FreeRTOS

Os arquivos de log do CloudTrail podem conter uma ou mais entradas de log. Cada entrada lista vários eventos com formatação JSON. Uma entrada de log representa uma única solicitação de qualquer origem e inclui informações sobre a ação solicitada, a data e hora da ação, parâmetros de solicitação, e assim por diante. As entradas de log não são um rastreamento de pilha ordenada das chamadas de API pública. Assim, elas não são exibidas em nenhuma ordem específica.

O exemplo a seguir mostra uma entrada de log do CloudTrail que demonstra o log de uma chamada à ação `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"
}
```

Obtenção de detalhes da falha CreateOTAUpdate usando a AWS CLI

Se o processo de criação de um trabalho de atualização OTA falhar, existem ações que você pode tomar para solucionar o problema. Quando você cria um trabalho de atualização OTA, o serviço de gerenciamento OTA cria um trabalho de IoT e o programa para os dispositivos de destino, além disso, esse processo também cria ou usa outros tipos de recursos da AWS em sua conta (um trabalho de assinatura de código, um fluxo do AWS IoT, um objeto do Amazon S3). Todo erro encontrado pode fazer o processo obter falha sem criar um trabalho do AWS IoT. Nesta seção de solução de problemas, fornecemos instruções sobre como recuperar os detalhes da falha.

1. Instale e configure a AWS CLI.
2. Execute `aws configure` e insira as seguintes informações.

```
$ 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 obter mais informações, consulte a Configuração rápida com `aws configure`.

3. Execute:

```
aws iot get-ota-update --ota-update-id ota_update_job_001
```

Em que *ota_update_job_001* é o ID que você deu à atualização OTA ao criá-la.

4. A saída será semelhante a esta:

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

Se a criação falhar, o campo `otaUpdateStatus` na saída do comando conterá `CREATE_FAILED` e o campo `errorInfo` conterá os detalhes da falha.

Obtenção de códigos de falha de OTA com a AWS CLI

1. Instale e configure a AWS CLI.
2. Execute `aws configure` e insira as seguintes informações.

```
$ 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 obter mais informações, consulte a Configuração rápida com `aws configure`.

3. Execute:

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

Em que *JobID* é a string de ID do trabalho completo para o trabalho cujo status queremos obter (estava associado ao trabalho de atualização OTA quando ele foi criado)e *ThingName* é o nome da coisa do AWS IoT com o qual o dispositivo está registrado no AWS IoT

4. A saída será semelhante a esta:

```
{
    "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
    }
}
```

Neste exemplo de saída, o "`reason`" no "`detailsmap`" tem dois campos: o campo mostrado como "0xEEEEEEEE" contém o código de erro genérico do agente de OTA; o campo mostrado como "0xffffffff" contém o subcódigo. Os códigos de erro genéricos estão listados em https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws__ota__agent_8h.html. Consulte os códigos de erro com o prefixo "`kOTA_Err_`". O subcódigo pode ser um código específico da plataforma ou fornecer mais detalhes sobre o erro genérico.

Solução de problemas de atualizações de OTA de vários dispositivos

Para executar OTAs em vários dispositivos (coisas) usando a mesma imagem de firmware, implemente uma função (por exemplo `getThingName()`) que recupera `clientcredentialIOT_THING_NAME` da memória não volátil. Verifique se essa função lê o nome da coisa de uma parte da memória não volátil que não é substituída pela OTA e se o nome da coisa é provisionado antes de executar o primeiro trabalho. Se você estiver usando o fluxo JITP, será possível ler o nome da coisa fora do nome comum do certificado do dispositivo.

Solução de problemas de atualizações de OTA com o Launchpad CC3220SF da Texas Instruments

A plataforma de software CC3220SF Launchpad fornece um mecanismo de detecção de adulteração de software. Ele usa um contador de alertas de segurança que é incrementado sempre que houver uma violação de integridade. O dispositivo é bloqueado quando o contador de alertas de segurança atinge um limite predeterminado (o padrão é 15) e o host recebe o evento assíncrono `SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT`. O dispositivo bloqueado, por sua vez, tem acessibilidade limitada. Para recuperar o dispositivo, você pode reprogramá-lo ou usar o processo de restaurar padrões de fábrica para reverter para a imagem de fábrica. É necessário programar o comportamento desejado atualizando o manipulador de eventos assíncrono em `network_if.c`.

Bibliotecas do FreeRTOS

As bibliotecas do FreeRTOS oferecem funcionalidade adicional para o kernel do FreeRTOS e suas bibliotecas internas. Você pode usar as bibliotecas do FreeRTOS para redes e segurança em aplicativos incorporados. As bibliotecas do FreeRTOS também habilitam os aplicativos a interagir com os serviços do AWS IoT. O FreeRTOS inclui bibliotecas que permitem:

- Conecte de forma segura dispositivos à nuvem da AWS IoT usando MQTT e sombras do dispositivo.
- Descubra e conecte-se a núcleos do AWS IoT Greengrass.
- Gerencie conexões de Wi-Fi.
- Ouça e processe Atualizações sem fios do FreeRTOS.

O diretório `libraries` contém o código-fonte das bibliotecas FreeRTOS. Há funções auxiliares úteis para implementar a funcionalidade da biblioteca. Não recomendamos que você altere essas funções auxiliares.

Bibliotecas de portabilidade do FreeRTOS

As bibliotecas de portabilidade a seguir estão inclusas nas configurações do FreeRTOS que estão disponíveis para download no console do FreeRTOS. Essas bibliotecas são dependentes de plataforma. O conteúdo muda de acordo com a plataforma de hardware. Para obter informações sobre a portabilidade dessas bibliotecas para um dispositivo, consulte o Guia de portabilidade do FreeRTOS.

Bibliotecas de portabilidade do FreeRTOS

Ferramentas	Referência da API	Descrição
Bluetooth Low Energy	Referência de API do Bluetooth Low Energy	Usando a biblioteca Bluetooth Low Energy do FreeRTOS, o microcont rolador pode se comunicar com o agente MQTT do AWS IoT por meio de um dispositivo de gateway. Para obter mais informações, consulte Biblioteca de Bluetooth Low Energy.
Atualizações remotas (OTA, Over-the-Air)	Referência de API de atualização sem fios de AWS IoT	A biblioteca de atualização sem fios de AWS IoT do FreeRTOS permite gerenciar notificações de atualização, baixar atualizações e realizar a verificação criptográfica das atualizações de firmware no dispositivo do FreeRTOS. Para obter mais informações, consulte Biblioteca sem fios do AWS IoT.
FreeRTOS+POSIX	Referência da API do FreeRTOS+ POSIX	É possível usar a biblioteca FreeRTOS+POSIX para transferir aplicativos compatíveis com POSIX para ecossistemas do FreeRTOS. Para obter mais informações, consulte FreeRTOS+POSIX.
Secure Sockets	Referência de API de Secure Sockets	Para obter mais informações, consulte Biblioteca de Secure Sockets.
FreeRTOS+TCP	Referência da API do FreeRTOS+ TCP	FreeRTOS+TCP é uma pilha TCP/ IP escalável, de código aberto e segura para o thread do FreeRTOS.

Ferramentas	Referência da API	Descrição
		Para obter mais informações, consulte FreeRTOS+TCP.
Wi-Fi	Referência de API do Wi-Fi	A biblioteca de Wi-Fi do FreeRTOS permite interagir com a pilha sem fio de baixo nível do microcontrolador. Para obter mais informações, consulte Biblioteca de Wi-Fi.
corePKCS11		A biblioteca corePKCS11 é uma implementação de referência do Padrão de criptografia de chave pública #11, para oferecer suporte ao provisionamento e à autentica ção do cliente TLS. Para obter mais informações, consulte Biblioteca corePKCS11.
TLS		Para obter mais informações, consulte Transport Layer Security.
E/S comum	Referência de API de E/S comum	Para obter mais informações, consulte E/S comum.
Cellular Interface	Referência de API da Cellular Interface	A biblioteca Cellular Interface expõe os recursos de alguns modems de rede celular populares por meio de uma API uniforme. Para obter mais informações, consulte Biblioteca Cellular Interface.

Bibliotecas de aplicativos do FreeRTOS

Opcionalmente, você pode incluir as seguintes bibliotecas de aplicativos independentes na configuração do FreeRTOS para interagir com serviços do AWS IoT na nuvem.

> **Note**
>
> Algumas das bibliotecas de aplicativo têm as mesmas APIs que as bibliotecas no AWS IoT Device SDK para C incorporado. Para essas bibliotecas, consulte a Referência de API em C do AWS IoT Device SDK. Para obter mais informações sobre AWS IoT Device SDK para C incorporado, consulte SDK de dispositivo da AWS IoT para C incorporado.

Bibliotecas de aplicativos do FreeRTOS

Ferramentas	Referência da API	Descrição
AWS IoT Device Defender	Referência de API para o SDK de C do Device Defender	A biblioteca AWS IoT Device Defender do FreeRTOS conecta seu dispositivo FreeRTOS ao AWS IoT Device Defender. Para obter mais informações, consulte Biblioteca de AWS IoT Device Defender.
AWS IoT Greengrass	Referência de API do Greengrass	A biblioteca AWS IoT Greengras s do FreeRTOS conecta seu dispositivo FreeRTOS ao AWS IoT Greengrass. Para obter mais informações, consulte Biblioteca de descoberta do AWS IoT Greengrass.
MQTT	Referência de API da biblioteca MQTT (v1.x.x) Referência de API do agente MQTT (v1) Referência de API para o SDK de C de MQTT (v2.x.x)	A biblioteca coreMQTT fornece um cliente para o dispositivo do FreeRTOS publicar e assinar tópicos do MQTT. MQTT é o protocolo usado pelos dispositivos para interagir com a AWS IoT. Para obter mais informações sobre a versão 3.0.0 da bibliotec

Ferramentas	Referência da API	Descrição
		a coreMQTT, consulte Biblioteca coreMQTT.
coreMQTT Agent	Referência de API da biblioteca coreMQTT Agent	A biblioteca coreMQTT Agent é uma API de alto nível que adiciona segurança de threads à biblioteca coreMQTT. Ela permite criar uma tarefa de agente MQTT dedicada que gerencia uma conexão MQTT em segundo plano e sem precisar de nenhuma intervenção de outras tarefas. A biblioteca fornece equivalentes seguros de threads às APIs do coreMQTT, para que possa ser usada em ambientes com threads múltiplos. Para obter mais informações sobre a biblioteca coreMQTT Agent, consulte Biblioteca coreMQTT Agent.
Device Shadow da AWS IoT	Referência de API para o SDK de C do device shadow	A biblioteca do AWS IoT Device Shadow permite que o dispositivo do FreeRTOS interaja com sombras de dispositivos do AWS IoT. Para obter mais informações, consulte Biblioteca de Device Shadow da AWS IoT.

Configuração das bibliotecas do FreeRTOS

As definições de configuração para FreeRTOS e AWS IoT Device SDK para C incorporado são definidas como constantes do pré-processador C. Você pode definir configurações com um arquivo de configuração global ou usando uma opção de compilador, como `-D` em `gcc`. Como as definições

de configuração são definidas como constantes de tempo de compilação, uma biblioteca deverá ser reconstruída se uma definição de configuração for alterada.

Se você quiser usar um arquivo de configuração global para definir as opções de configuração, crie e salve o arquivo com o nome `iot_config.h`, e coloque-o no seu caminho de inclusão. No arquivo, use diretivas `#define` para configurar as bibliotecas, demonstrações e testes do FreeRTOS.

Para obter mais informações sobre as opções de configuração global compatíveis, consulte a Referência do arquivo de configuração global.

Biblioteca backoffAlgorithm

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

A biblioteca backoffAlgorithm é uma biblioteca utilitária usada para espaçar retransmissões repetidas do mesmo bloco de dados, para evitar o congestionamento da rede. Essa biblioteca calcula o período de recuo para repetir as operações de rede (como uma falha na conexão de rede com o servidor) usando um algoritmo de recuo exponencial com tremulação.

O recuo exponencial com tremulação é normalmente usado ao tentar novamente uma falha na conexão ou solicitação de rede em um servidor causada pelo congestionamento da rede ou devido à cargas altas no servidor. Ele é usado para distribuir o tempo das solicitações de repetição criadas por vários dispositivos que tentam se conectar à rede ao mesmo tempo. Em um ambiente com pouca conectividade, um cliente pode se desconectar a qualquer momento; portanto, uma estratégia de recuo também ajuda o cliente a economizar bateria ao não tentar reconexões repetidamente quando é improvável que elas tenham êxito.

A biblioteca é escrita em C e criada para ser compatível com ISO C90 e MISRA C:2012. A biblioteca não depende de nenhuma biblioteca adicional além da biblioteca C padrão e não tem alocação de heap, o que a torna adequada para microcontroladores do IoT, mas também totalmente portável para outras plataformas.

Essa biblioteca pode ser usada gratuitamente e é distribuída sob a licença de código aberto do MIT.

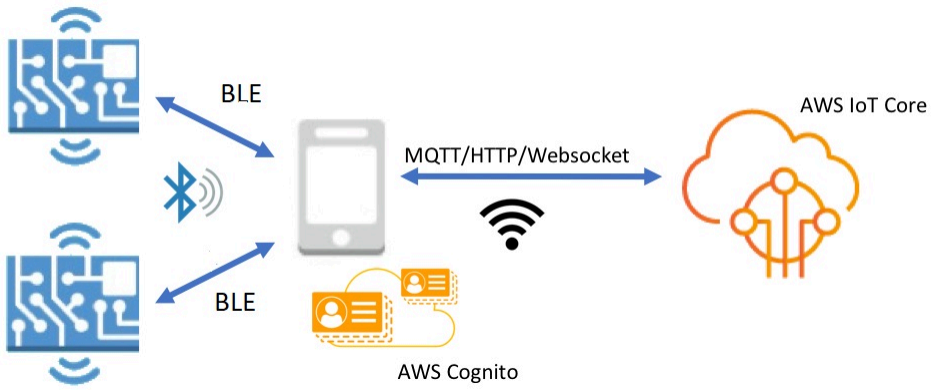
Tamanho de código de backoffAlgorithm (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
backoff_algorithm.c	0,1 K	0,1 K
Estimativas totais	0,1 K	0,1 K

Biblioteca de Bluetooth Low Energy

> **Important**
>
> Essa biblioteca está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Visão geral

O FreeRTOS é compatível com a publicação e inscrição em tópicos do Message Queuing Telemetry Transport (MQTT) por Bluetooth Low Energy por meio de um dispositivo de proxy, como um celular. Com a biblioteca FreeRTOS Bluetooth Low Energy (BLE), seu microcontrolador pode se comunicar com segurança com o broker MQTT. AWS IoT

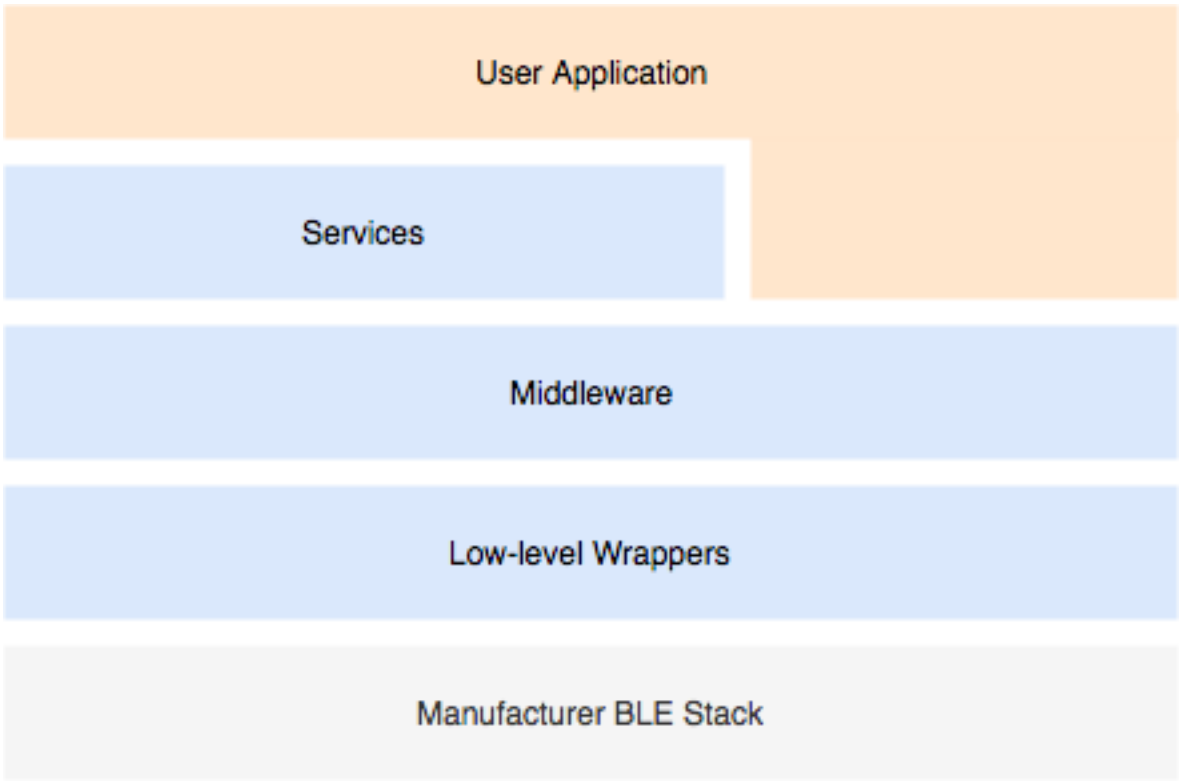


Usando os SDKs móveis de dispositivos Bluetooth do FreeRTOS, grave aplicações móveis nativas que se comunicam com as aplicações incorporadas no seu microcontrolador por BLE. Para obter mais informações sobre os SDKs móveis, consulte SDKs móveis para dispositivos Bluetooth do FreeRTOS.

A biblioteca BLE do FreeRTOS inclui serviços para configurar redes Wi-Fi, transferir grandes quantidades de dados e fornecer abstrações de rede pela BLE. A biblioteca BLE do FreeRTOS também inclui middleware e APIs de nível inferior para controle mais direto sobre a pilha BLE.

Arquitetura

Três camadas compõem a biblioteca BLE do FreeRTOS serviços, middleware e wrappers de nível inferior.



Serviços

A camada de serviços BLE do FreeRTOS consiste em quatro serviços de recursos genéricos (GATT) que utilizam as APIs de middleware:

- Informações do dispositivo
- Provisionamento de Wi-Fi
- Abstração de rede
- Transferência de objetos grandes

Informações do dispositivo

O serviço de informações do dispositivo reúne informações sobre o microcontrolador, incluindo:

- A versão do FreeRTOS que seu dispositivo está usando.
- O AWS IoT endpoint da conta na qual o dispositivo está registrado.
- Unidade de transmissão máxima de Bluetooth Low Energy (MTU).

Provisionamento de Wi-Fi

O serviço de provisionamento de Wi-Fi permite que os microcontroladores com recursos de Wi-Fi façam o seguinte:

- Listar redes no intervalo.
- Salvar redes e credenciais de rede na memória flash.
- Definir a prioridade da rede.
- Excluir redes e credenciais de rede da memória flash.

Abstração de rede

O serviço de abstração de rede abstrai o tipo de conexão de rede para aplicações. Uma API comum interage com a pilha de hardware Wi-Fi, Ethernet e Bluetooth Low Energy do seu dispositivo, permitindo que uma aplicação seja compatível com vários tipos de conexão.

Transferência de objetos grandes

O serviço de transferência de objetos grandes envia para e recebe dados de um cliente. Outros serviços, como provisionamento de Wi-Fi e abstração de rede, usam o serviço de transferência de objetos grandes para enviar e receber dados. Você também pode usar a API de transferência de objetos grandes para interagir com o serviço diretamente.

MQTT por BLE

MQTT por BLE contém o perfil GATT para criar um serviço proxy MQTT por BLE. O serviço proxy MQTT permite que um cliente MQTT se comunique com o agente AWS MQTT por meio de um dispositivo de gateway. Por exemplo, você pode usar o serviço de proxy para conectar um dispositivo executando FreeRTOS ao MQTT por meio de um AWS aplicativo de smartphone. O dispositivo BLE é o servidor GATT e expõe serviços e características do dispositivo gateway. O servidor GATT usa esses serviços e características expostos para realizar operações MQTT com a nuvem desse dispositivo. Para obter mais detalhes, consulte Apêndice A: perfil MQTT por GATT BLE .

Middleware

O middleware da biblioteca Bluetooth Low Energy do FreeRTOS é uma abstração de APIs de nível inferior. As APIs de middleware compõem uma interface mais amigável para a pilha Bluetooth Low Energy.

Usando APIs de middleware, você pode registrar vários retornos de chamada, em várias camadas, para um único evento. Iniciar o middleware da biblioteca Bluetooth Low Energy também inicializa serviços e começa a publicidade.

Assinatura de retorno de chamada flexível

Suponha que o hardware Bluetooth Low Energy se desconecta, e o serviço MQTT por Bluetooth Low Energy precisa detectar a desconexão. Uma aplicação gravada também poderá ter que detectar o mesmo evento de desconexão. O middleware Bluetooth Low Energy pode rotear o evento para diferentes partes do código onde os retornos de chamada foram registrados, sem fazer com que as camadas mais altas compitam por recursos de nível inferior.

Wrappers de baixo nível

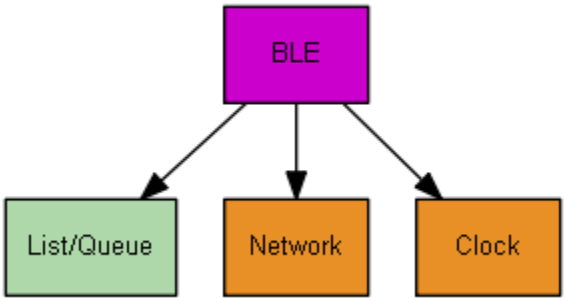
Os wrappers de nível baixo da Bluetooth Low Energy do FreeRTOS são uma abstração da pilha do fabricante Bluetooth Low Energy. Os wrappers de baixo nível oferecem um conjunto comum de APIs para controle direto sobre o hardware. As APIs de baixo nível otimizam o uso de RAM, mas são limitadas em funcionalidade.

Use as APIs de serviço da Bluetooth Low Energy para interagir com os serviços da Bluetooth Low Energy. As APIs de serviço exigem mais recursos do que as APIs de baixo nível.

Dependências e requisitos

A biblioteca Bluetooth Low Energy tem as seguintes dependências diretas:

- Biblioteca Linear Containers
- Uma camada de plataforma que faz interface com o sistema operacional para o gerenciamento de threads, temporizadores, funções de relógio e acesso à rede.



Somente o serviço de provisionamento de Wi-Fi tem dependências de biblioteca do FreeRTOS:

Serviço de GATT	Dependência
Provisionamento de Wi-Fi	Biblioteca de Wi-Fi

Para se comunicar com o corretor AWS IoT MQTT, você deve ter uma AWS conta e registrar seus dispositivos como AWS IoT coisas. Para obter mais informações sobre a configuração, consulte o Guia do desenvolvedor do AWS IoT.

O Bluetooth Low Energy do FreeRTOS usa o Amazon Cognito para a autenticação do usuário em seu dispositivo móvel. Para usar os serviços de proxy MQTT, é necessário criar uma identidade do Amazon Cognito e grupos de usuários. Cada identidade do Amazon Cognito deve ter a política adequada anexada. Para obter mais informações, consulte o Guia do desenvolvedor do Amazon Cognito.

Arquivo de configuração da biblioteca

As aplicações que usam o serviço MQTT do FreeRTOS pela Bluetooth Low Energy devem fornecer um arquivo de cabeçalho `iot_ble_config.h`, no qual são definidos parâmetros de configuração. Os parâmetros de configuração não definidos assumem os valores padrão especificados em `iot_ble_config_defaults.h`.

Alguns parâmetros de configuração importantes incluem:

`IOT_BLE_ADD_CUSTOM_SERVICES`

Permite que os usuários criem seus próprios serviços.

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

Permite que os usuários personalizem o anúncio e verifiquem as mensagens de resposta.

Para obter mais informações, consulte Referência de API do Bluetooth Low Energy.

Otimização

Ao otimizar o desempenho da placa, considere o seguinte:

- APIs de baixo nível usam menos RAM, mas oferecem funcionalidade ilimitada.
- Você pode definir o parâmetro `bleconfigMAX_NETWORK` no arquivo de cabeçalho `iot_ble_config.h` como um valor menor para reduzir a quantidade consumida da pilha.
- Você pode aumentar o tamanho da MTU até seu valor máximo para limitar o armazenamento em buffer de mensagens e fazer com que o código execute mais rápido e consuma menos RAM.

Restrições de uso

Por padrão, a biblioteca Bluetooth Low Energy do FreeRTOS define a propriedade `eBTpropertySecureConnectionOnly` como VERDADEIRA, que coloca o dispositivo em um modo Somente conexões seguras. Conforme especificado pela Bluetooth Core Specification v5.0, Vol. 3, Parte C, 10.2.4, quando um dispositivo está no modo Somente conexões seguras, é necessário o nível de segurança LE mais alto do modo 1, o nível 4, para acessar qualquer atributo que tem permissões mais altas que o nível de segurança LE mais baixo do modo 1, o nível 1. No nível 4 de segurança LE do modo 1, um dispositivo deve ter recursos de entrada e saída para comparação numérica.

Aqui estão os modos compatíveis e suas propriedades associadas:

Modo 1, Nível 1 (sem segurança)

```
/* 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, Nível 2 (emparelhamento com criptografia não autenticada)

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON       ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                    ( eBTIONone )
```

Modo 1, Nível 3 (emparelhamento com criptografia autenticada)

Esse modo não é compatível.

Modo 1, Nível 4 (emparelhamento com criptografia de conexões seguras LE autenticadas)

Esse modo é suportado por padrão.

Para obter informações sobre os modos de segurança LE, consulte a Bluetooth Core Specification v5.0, Vol. 3, Parte C, 10.2.1.

Inicialização

Se a aplicação interage com a pilha Bluetooth Low Energy por meio de middleware, só é necessário inicializar o middleware. O middleware cuida da inicialização das camadas inferiores da pilha.

Middleware

Para inicializar o middleware

1. Inicialize qualquer driver de hardware Bluetooth Low Energy antes de chamar a API de middleware da Bluetooth Low Energy.
2. Habilitar Bluetooth Low Energy.
3. Inicialize o middleware com `IotBLE_Init()`.

> **Note**
>
> Essa etapa de inicialização não é necessária se você estiver executando as AWS demonstrações. A inicialização de demonstração é processada pelo gerenciador de rede, localizado em *`freertos`*`/demos/network_manager`.

APIs de baixo nível

Se não quiser usar os serviços de GATT Bluetooth Low Energy do FreeRTOS, é possível ignorar o middleware e interagir diretamente com as APIs de baixo nível para economizar recursos.

Para inicializar as APIs de baixo nível

1. Inicialize qualquer driver de hardware Bluetooth Low Energy antes de chamar as APIs. A inicialização de driver não faz parte das APIs de baixo nível da Bluetooth Low Energy.

2. A API de baixo nível da Bluetooth Low Energy fornece uma chamada de habilitação/desabilitação para a pilha Bluetooth Low Energy a fim de otimizar a potência e os recursos. Antes de chamar as APIs, habilite Bluetooth Low Energy.

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. O gerenciador de Bluetooth contém APIs que são comuns à Bluetooth Low Energy e ao Bluetooth clássico. Os retornos de chamada para o gerenciador comum devem ser inicializados depois.

```
xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
```

4. O adaptador Bluetooth Low Energy se encaixa em cima da API comum. É necessário inicializar os retornos de chamada da mesma forma que a API comum foi inicializada.

```
xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * )
 xBTInterface.pxBTInterface->pxGetLeAdapter();
xStatus = xBTInterface.pxBTLeAdapterInterface-
>pxBleAdapterInit( &xBTBleAdapterCb );
```

5. Registre a nova aplicação do usuário.

```
xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
```

6. Inicialize os retornos de chamada para os servidores GATT.

```
xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
 xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

Depois de inicializar o adaptador Bluetooth Low Energy, você pode adicionar um servidor GATT. Só é possível registrar um servidor GATT por vez.

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. Defina as propriedades da aplicação, como somente conexão segura e tamanho da MTU.

```
xStatus = xBTInterface.pxBTInterface-
>pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

Referência de API

Para obter uma referência completa de API, consulte Referência de API do Bluetooth Low Energy.

Exemplo de uso

Os exemplos a seguir demonstram como usar a biblioteca Bluetooth Low Energy para publicidade e criação de novos serviços. Para obter aplicações de demonstração da Bluetooth Low Energy do FreeRTOS completos, consulte Aplicações de demonstração de Bluetooth Low Energy.

Publicidade

1. Na aplicação, defina o UUID de publicidade:

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBTuuidType128
};
```

2. Em seguida, defina a função de retorno de chamada `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;
}
```

Esse retorno de chamada envia o UUID na mensagem de anúncio e o nome completo na resposta de verificação.

3. Abra `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` e defina `IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG` como 1. Isso dispara o retorno de chamada `IotBle_SetCustomAdvCb`.

Adição de um novo serviço

Para ver exemplos completos de serviços, consulte *freertos*`/.../ble/services`.

1. Crie UUIDs para a característica do serviço e descritores:

```
#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. Crie um buffer para registrar as alças da característica e dos descritores:

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. Crie a tabela de atributo. Para economizar RAM, defina a tabela como `const`.

> **Important**
>
> Sempre crie os atributos em ordem, com o serviço como o primeiro 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. Crie uma matriz de retornos de chamada. Essa matriz de retornos de chamada deve seguir a mesma ordem que a matriz de tabela definida acima.

Por exemplo, se `vReadCounter` é acionado quando `xCharCounterUUID_TYPE` é acessado, e `vWriteCommand` é acionado quando `xCharControlUUID_TYPE` é acessado, defina a matriz da seguinte forma:

```
static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes]
 =
    {
  NULL,
  vReadCounter,
  vEnableNotification,
  vWriteCommand
};
```

5. Crie o serviço:

```
static const BTService_t xGattDemoService =
{
  .xNumberOfAttributes = egattDemoNbAttributes,
  .ucInstId = 0,
  .xType = eBTServiceTypePrimary,
  .pusHandlesBuffer = usHandlesBuffer,
  .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. Chame a API `IotBle_CreateService` com a estrutura que você criou na etapa anterior. O middleware sincroniza a criação de todos os serviços e, portanto, é necessário que todos os novos serviços já tenham sido definidos quando o `IotBle_AddCustomServicesCb` retorno de chamada for acionado.

   a. Defina `IOT_BLE_ADD_CUSTOM_SERVICES` como 1 em `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h`.

   b. Crie IotBle _ AddCustomServicesCb em seu aplicativo:

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

Portabilidade

Periférico de entrada e saída do usuário

Uma conexão segura requer entrada e saída para comparação numérica. O evento `eBLENumericComparisonCallback` pode ser registrado usando o gerenciador de eventos:

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

O periférico deve exibir a chave de acesso numérica e assumir o resultado da comparação como uma entrada.

Transferência de implementações da API

Para fazer a portabilidade do FreeRTOS para um novo destino, é necessário implementar algumas APIs para o serviço de Provisionamento de Wi-Fi e funcionalidade de Bluetooth Low Energy.

APIs de Bluetooth Low Energy

Para usar o middleware Bluetooth Low Energy do FreeRTOS, é necessário implementar algumas APIs.

APIs comuns entre GAP para Bluetooth Classic e GAP para Bluetooth Low Energy

- `pxBtManagerInit`
- `pxEnable`
- `pxDisable`
- `pxGetDeviceProperty`
- `pxSetDeviceProperty` (Todas as opções são obrigatórias, exceto `eBTpropertyRemoteRssi` e `eBTpropertyRemoteVersionInfo`)
- `pxPair`
- `pxRemoveBond`
- `pxGetConnectionState`
- `pxPinReply`
- `pxSspReply`

- pxGetTxpower
- pxGetLeAdapter
- pxDeviceStateChangedCb
- pxAdapterPropertiesCb
- pxSspRequestCb
- pxPairingStateChangedCb
- pxTxPowerCb

APIs específicas para GAP para Bluetooth Low Energy

- pxRegisterBleApp
- pxUnregisterBleApp
- pxBleAdapterInit
- pxStartAdv
- pxStopAdv
- pxSetAdvData
- pxConnParameterUpdateRequest
- pxRegisterBleAdapterCb
- pxAdvStartCb
- pxSetAdvDataCb
- pxConnParameterUpdateRequestCb
- pxCongestionCb

Servidor 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 obter mais informações sobre a portabilidade da biblioteca Bluetooth Low Energy do FreeRTOS para sua plataforma, consulte Portabilidade da biblioteca Bluetooth Low Energy no Guia de portabilidade do FreeRTOS.

SDKs móveis para dispositivos Bluetooth do FreeRTOS

> **Important**
>
> Essa biblioteca está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Você pode usar os SDKs móveis de dispositivos Bluetooth do FreeRTOS para criar aplicações móveis que interagem com o microcontrolador pela Bluetooth Low Energy. Os SDKs móveis também podem se comunicar com AWS serviços usando o Amazon Cognito para autenticação de usuários.

SDK do Android para dispositivos Bluetooth do FreeRTOS

Use o SDK para Android de dispositivos Bluetooth do FreeRTOS para criar aplicações móveis do Android que interagem com o microcontrolador pela Bluetooth Low Energy. O SDK está disponível em. GitHub

Para instalar o SDK do Android para dispositivos Bluetooth FreeRTOS, siga as instruções para "Configurar o SDK" no arquivo README.md do projeto.

Para obter informações sobre como configurar e executar a aplicação móvel de demonstração incluído com o SDK, consulte Pré-requisitos e aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS.

SDK do iOS para dispositivos Bluetooth do FreeRTOS

Use o SDK para iOS de dispositivos Bluetooth do FreeRTOS para criar aplicações móveis do iOS que interagem com o microcontrolador pela Bluetooth Low Energy. O SDK está disponível em. GitHub

Para instalar o SDK para iOS

1. Instale CocoaPods:

```
$ gem install cocoapods
$ pod setup
```

> **Note**
>
> Talvez seja necessário usar `sudo` para instalar CocoaPods.

2. Instale o SDK com CocoaPods (adicione isso ao seu podfile):

```
$ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
```

Para obter informações sobre como configurar e executar a aplicação móvel de demonstração incluído com o SDK, consulte Pré-requisitos e aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS.

Apêndice A: perfil MQTT por GATT BLE

Detalhes do serviço GATT

O MQTT por BLE usa uma instância do serviço GATT de transferência de dados para enviar mensagens de representação concisa de objetos binários (CBOR) do MQTT entre o dispositivo FreeRTOS e o dispositivo proxy. O serviço de transferência de dados expõe certas características que ajudam a enviar e receber dados brutos pelo protocolo GATT da BLE. Ele também lida com a fragmentação e montagem de cargas úteis maiores que o tamanho da unidade de transferência máxima (MTU) BLE.

UUID de serviço

`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00`

Instâncias de serviço

Uma instância do serviço GATT é criada para cada sessão do MQTT com o agente. Cada serviço tem um UUID exclusivo (dois bytes) que identifica o tipo. Cada instância individual é diferenciada pelo ID da instância.

Cada serviço é instanciado como um serviço primário em cada dispositivo de servidor BLE. Você pode criar várias instâncias do serviço em um determinado dispositivo. O tipo de serviço proxy MQTT tem um UUID exclusivo.

Características

Formato característico de conteúdo: CBOR

Tamanho máximo do valor característico: 512 bytes

Característica	Requisito	Propriedades obrigatórias	Propriedades opcionais	Permissões de segurança	Breve descrição	UUID
Controle	M	Escrever	Nenhum	Gravação precisa de criptografia	Usado para iniciar e interromper o proxy MQTT.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF01
TXMessage	M	Leitura, notificação	Nenhum	Leitura precisa de criptografia	Usado para enviar uma notificação contendo uma mensagem para um agente por meio de um proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF02
RXMessage	M	Leitura, gravação sem resposta	Nenhum	Leitura, gravação precisa de criptografia	Usado para receber uma mensagem de um corretor por meio	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF03

Característica	Requisito	Propriedades obrigatórias	Propriedades opcionais	Permissões de segurança	Breve descrição	UUID
					de um proxy.	
TX LargeMessage	M	Leitura, notificação	Nenhum	Leitura precisa de criptografia	Usado para enviar uma grande mensagem (Mensagem > Tamanho do MTU da BLE) para um agente por meio de um proxy.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF04`

Característica	Requisito	Propriedades obrigatórias	Propriedades opcionais	Permissões de segurança	Breve descrição	UUID
RX LargeMessage	M	Leitura, gravação sem resposta	Nenhum	Leitura, gravação precisa de criptografia	Usado para receber uma grande mensagem (Mensagem > Tamanho do MTU da BLE) de um agente por meio de um proxy.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF05`

Requisitos de procedimento do GATT

Valores característicos de leitura	Obrigatório
Valores característicos longos de leitura	Obrigatório
Valores característicos de gravação	Obrigatório
Valores característicos longos de gravação	Obrigatório
Descritores característicos de leitura	Obrigatório
Descritores característicos de gravação	Obrigatório
Notificações	Obrigatório

Indicações	Obrigatório

Tipos de mensagem

Os tipos de mensagem a seguir são trocados.

Tipo de mensagem	Message	Mapa com esses pares de chave/valor
0x01	CONECTAR	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (1) • Chave = "d", valor = tipo 3, string de texto, identific ador do cliente para a sessão • Chave = "a", valor = tipo 3, string de texto, endpoint do agente para a sessão • Chave = "c", valor = tipo de valor simples verdadeiro/ falso
0x02	CONNACK	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (2) • Chave = "s", valor = tipo 0 inteiro, código de status
0x03	PUBLISH	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (3) • Chave = "u", valor = tipo 3, string de texto, tópico para publicação

Tipo de mensagem	Message	Mapa com esses pares de chave/valor
		• Chave = "n", valor = tipo 0, inteiro, QoS para publicação • Chave = "i", valor = tipo 0, número inteiro, identificador de mensagem, somente para publicações de QoS 1 • Chave = "k", valor = tipo 2, string de bytes, carga útil para publicação
0x04	PUBACK	• Enviado somente para mensagens de QoS 1. • Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (4) • Chave = "i", valor = tipo 0, inteiro, identificador de mensagem
0x08	SUBSCRIBE	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (8) • Chave = "v", valor = tipo 4, matriz de strings de texto, tópicos para assinatura • Chave = "o", valor = tipo 4, matriz de números inteiros, QoS para assinatura • Chave = "i", valor = tipo 0, inteiro, identificador de mensagem

Tipo de mensagem	Message	Mapa com esses pares de chave/valor
0x09	SUBACK	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (9) • Chave = "i", valor = tipo 0, inteiro, identificador de mensagem • Chave = "s", valor = tipo 0 inteiro, código de status para assinatura
0X0A	CANCELAR INSCRIÇÃO	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (10) • Chave = "v", valor = tipo 4, matriz de strings de texto, tópicos para cancelamento de assinatura • Chave = "i", valor = tipo 0, inteiro, identificador de mensagem
0x0B	UNSUBACK	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (11) • Chave = "i", valor = tipo 0, inteiro, identificador de mensagem • Chave = “s”, valor = Tipo 0, número inteiro, código de status para UnSubscription

Tipo de mensagem	Message	Mapa com esses pares de chave/valor
0X0C	PINGREQ	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (12)
0x0D	PINGRESP	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (13)
0x0E	DISCONNNECT	• Chave = "w", valor = tipo 0 inteiro, tipo de mensagem (14)

Características de transferência de carga útil grande

TX LargeMessage

O TX LargeMessage é usado pelo dispositivo para enviar uma grande carga útil maior que o tamanho da MTU negociado para a conexão BLE.

- O dispositivo envia os primeiros bytes de MTU da carga como uma notificação por meio da característica.
- O proxy envia uma solicitação de leitura sobre essa característica para os bytes restantes.
- O dispositivo envia até o tamanho da MTU ou os bytes restantes da carga, o que for menor. Cada vez, ele aumenta o deslocamento lido pelo tamanho da carga útil enviada.
- O proxy continuará lendo a característica até obter uma carga útil de comprimento zero ou uma carga útil menor que o tamanho da MTU.
- Se o dispositivo não receber uma solicitação de leitura dentro de um tempo limite especificado, a transferência falhará e o proxy e o gateway liberarão o buffer.
- Se o proxy não receber uma solicitação de leitura dentro de um tempo limite especificado, a transferência falhará e o proxy liberará o buffer.

RX LargeMessage

O RX LargeMessage é usado pelo dispositivo para receber uma grande carga útil maior que o tamanho da MTU negociado para a conexão BLE.

- O proxy grava mensagens, até o tamanho da MTU, uma por uma, usando gravação com resposta nessa característica.
- O dispositivo armazena a mensagem em buffer até receber uma solicitação de gravação com tamanho zero ou menor que o tamanho da MTU.
- Se o dispositivo não receber uma solicitação de gravação dentro de um tempo limite especificado, a transferência falhará e o dispositivo liberará o buffer.
- Se o proxy não receber uma solicitação de gravação dentro de um tempo limite especificado, a transferência falhará e o proxy liberará o buffer.

Biblioteca Cellular Interface

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

A biblioteca Cellular Interface implementa uma API unificada simples que oculta a complexidade dos comandos AT específicos do modem de rede celular e expõe uma interface semelhante a um soquete para programadores C.

A maioria dos modems de rede celular implementa mais ou menos dos comandos AT definidos pelo padrão 3GPP TS v27.007. Este projeto fornece uma implementação desses comandos AT padrão em um componente comum reutilizável. Todas as três bibliotecas Cellular Interface deste projeto aproveitam esse código comum. A biblioteca de cada modem implementa apenas os comandos AT específicos do fornecedor e, em seguida, expõe a API completa da biblioteca Cellular Interface.

O componente comum que implementa o padrão 3GPP TS v27.007 foi escrito em conformidade com os seguintes critérios de qualidade de código:

- As pontuações de complexidade do GNU não são superiores a 8
- Padrão de codificação MISRA C:2012. Todo desvio do padrão é documentado nos comentários do código-fonte marcados por "coverity".

Dependências e requisitos

Não há dependência direta da biblioteca Cellular Interface. No entanto, Ethernet, Wi-Fi e rede celular não podem coexistir na pilha de rede do FreeRTOS. Os desenvolvedores devem escolher uma das interfaces de rede para integrar com a biblioteca Secure Sockets.

Portabilidade

Para obter mais informações sobre a portabilidade da biblioteca Cellular Interface para sua plataforma, consulte Portabilidade da biblioteca Cellular Interface no Guia de portabilidade do FreeRTOS.

Uso de memória

Tamanho de código da biblioteca Cellular Interface (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -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
Estimativas totais	13,9 K	12,4 K

Conceitos básicos

Fazer download do código-fonte

O código-fonte pode ser baixado como parte das bibliotecas do FreeRTOS ou individualmente.

Para clonar a biblioteca do Github usando HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Usando SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Estrutura de pastas

Na raiz desse repositório, você verá estas pastas:

- `source`: código comum reutilizável que implementa os comandos AT padrão definidos pelo 3GPP TS v27.007
- `doc`: documentação
- `test`: teste unitário e cbmc
- `tools`: ferramentas para análise estática da Coverity e CMock

Configurar e compilar a biblioteca

A biblioteca Cellular Interface deve ser compilada como parte de um aplicativo.
Para fazer isso, você deve fornecer determinadas configurações. O projeto FreeRTOS_Cellular_Interface_Windows_Simulator fornece um exemplo de como configurar a compilação. Mais informações podem ser encontradas nas Referências de API de rede celular.

Consulte a página Cellular Interface para obter mais informações.

Integre a biblioteca Cellular Interface com plataformas MCU

A biblioteca Cellular Interface é executada em MCUs usando uma interface abstrata, a Interface de comunicação, para se comunicar com modems de rede celular. Uma interface de comunicação também deve ser implementada na plataforma MCU. As implementações mais comuns da interface de comunicação se comunicam por meio de hardware UART, mas também podem ser implementadas em outras interfaces físicas, como SPI. A documentação da Interface de comunicação pode ser encontrada nas Referências de API da biblioteca de rede celular. Os seguintes exemplos de implementações da Interface de comunicação estão disponíveis:

- Interface de comunicação do simulador Windows do FreeRTOS
- Interface de comunicação UART de E/S comum do FreeRTOS

- Interface de comunicação da placa de descoberta STM32 L475
- Interface de comunicação da placa Sierra Sensor Hub

E/S comum

> **Important**
>
> Essa biblioteca está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Visão geral

Em geral, os drivers de dispositivo são independentes do sistema operacional subjacente e são específicos de uma determinada configuração de hardware. Uma camada de abstração de hardware (HAL) fornece uma interface comum entre drivers e código de aplicativo de nível superior. A HAL abstrai os detalhes de como um driver específico funciona e fornece uma API uniforme para controlar esses dispositivos. Você pode usar as mesmas APIs para acessar vários drivers de dispositivo em placas de referência baseadas em múltiplos microcontroladores (MCU).

A E/S comum do FreeRTOS atua como essa camada de abstração de hardware. Ela fornece um conjunto de APIs padrão para acessar dispositivos seriais comuns em placas de referência compatíveis. Essas APIs comuns se comunicam e interagem com esses periféricos e permitem que seu código funcione entre plataformas. Sem a E/S comum, escrever código para que funcione com dispositivos de baixo nível é uma tarefa específica do fornecedor de silício.

Periféricos compatíveis

- UART
- SPI
- I2C

Recursos compatíveis

- Leitura/gravação síncrona: a função não retorna até que a quantidade solicitada de dados seja transferida.

- Leitura/gravação assíncrona: a função retorna imediatamente e a transferência de dados ocorre de forma assíncrona. Quando a ação for concluída, um retorno de chamada de usuário registrado é chamado.

Específico do periférico

- I2C: combina várias operações em uma transação. Usado para realizar as ações de gravação e leitura em uma transação.
- SPI: transfere dados entre primário e secundário, o que significa que a gravação e a leitura ocorrem simultaneamente.

Portabilidade

Para obter mais informações, consulte o Guia de portabilidade do FreeRTOS.

Biblioteca de AWS IoT Device Defender

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

É possível usar a biblioteca do AWS IoT Device Defender para enviar métricas de segurança dos dispositivos de IoT para o AWS IoT Device Defender. Você pode usar o AWS IoT Device Defender para monitorar essas métricas de segurança dos dispositivos continuamente em busca de desvios do que você definiu como comportamento apropriado para cada dispositivo. Se algo parecer errado, o AWS IoT Device Defender enviará um alerta para que você possa tomar medidas para corrigir o problema. As interações com o AWS IoT Device Defender usam o MQTT, um protocolo leve de publicação e assinatura. Essa biblioteca fornece uma API para compor e reconhecer as strings de tópicos do MQTT usadas pelo AWS IoT Device Defender.

Para obter mais informações, consulte AWS IoT Device Defender no Guia do desenvolvedor do AWS IoT.

A biblioteca é escrita em C e criada para ser compatível com ISO C90 e MISRA C:2012. A biblioteca não depende de bibliotecas adicionais além da biblioteca C padrão. Ela também não depende

de plataformas, como threading ou sincronização. Ela pode ser usada com toda biblioteca MQTT e biblioteca JSON ou CBOR. A biblioteca tem provas que mostram o uso seguro da memória e a ausência de alocação de heap e isso a torna adequada para microcontroladores do IoT, mas também totalmente portável para outras plataformas.

A biblioteca do AWS IoT Device Defender pode ser usada gratuitamente e é distribuída sob a licença de código aberto do MIT.

Tamanho de código do AWS IoT Device Defender (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
defender.c	1,1 K	0,6 K
Estimativas totais	1,1 K	0,6 K

Biblioteca de descoberta do AWS IoT Greengrass

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Visão geral

A biblioteca de descobertas do AWS IoT Greengrass é usada pelos dispositivos do microcontrolador para descobrir um núcleo do Greengrass na rede. Usando as APIs de descoberta do AWS IoT Greengrass, o dispositivo pode enviar mensagens para um núcleo do Greengrass depois de encontrar o endpoint do núcleo.

Dependências e requisitos

Para usar a biblioteca de descoberta do Greengrass, é necessário criar um item na AWS IoT, incluindo um certificado e uma política. Para obter mais informações, consulte Conceitos básicos do AWS IoT.

É necessário definir valores para as constantes a seguir no arquivo *freertos*/demos/include/aws_clientcredential.h:

clientcredentialMQTT_BROKER_ENDPOINT

O endpoint da AWS IoT.

clientcredentialIOT_THING_NAME

O nome da coisa da IoT.

clientcredentialWIFI_SSID

O SSID da rede Wi-Fi.

clientcredentialWIFI_PASSWORD

A senha de Wi-Fi.

clientcredentialWIFI_SECURITY

O tipo de segurança usado pela rede Wi-Fi.

Também é necessário definir valores para as constantes a seguir no arquivo *freertos*/demos/include/aws_clientcredential_keys.h:

keyCLIENT_CERTIFICATE_PEM

O certificado PEM associado à coisa.

keyCLIENT_PRIVATE_KEY_PEM

A chave privada PEM associada à coisa.

É necessário ter um grupo do Greengrass e um dispositivo de núcleo configurados no console. Para obter mais informações, consulte o tópico Conceitos básicos sobre a AWS IoT Greengrass.

Embora a biblioteca coreMQTT não seja necessária para a conectividade do Greengrass, recomendamos que você a instale. A biblioteca pode ser usada para se comunicar com o núcleo do Greengrass após ele ter sido descoberto.

Referência de API

Para obter uma referência completa de API, consulte Referência de API do Greengrass.

Exemplo de uso

Fluxo de trabalho do Greengrass

O dispositivo MCU inicia o processo de descoberta solicitando à AWS IoT um arquivo JSON que contenha os parâmetros de conectividade de núcleo do Greengrass. Existem dois métodos para recuperar os parâmetros de conectividade de núcleo do Greengrass a partir do arquivo JSON:

- A seleção automática itera todos os núcleos do Greengrass listados no arquivo JSON e se conecta ao primeiro disponível.
- A seleção manual usa as informações em `aws_ggd_config.h` para se conectar ao núcleo especificado do Greengrass.

Como usar a API do Greengrass

Todas as opções de configuração padrão para a API do Greengrass são definidas em `aws_ggd_config_defaults.h`.

Se apenas um núcleo do Greengrass estiver presente, chame `GGD_GetGGCIPandCertificate` para solicitar o arquivo JSON com as informações de conectividade de núcleo do Greengrass. Quando `GGD_GetGGCIPandCertificate` é retornado, o parâmetro `pcBuffer` contém o texto do arquivo JSON. O parâmetro `pxHostAddressData` contém o endereço IP e a porta do núcleo do Greengrass aos quais você pode se conectar.

Para obter mais opções de personalização, como a alocação dinâmica de certificados, é necessário chamar as APIs a seguir:

`GGD_JSONRequestStart`

Faz uma solicitação HTTP GET para a AWS IoT para iniciar a solicitação de descoberta para descobrir um núcleo do Greengrass. `GD_SecureConnect_Send` é usado para enviar a solicitação à AWS IoT.

`GGD_JSONRequestGetSize`

Obtém o tamanho do arquivo JSON da resposta HTTP.

`GGD_JSONRequestGetFile`

Obtém a string do objeto JSON. `GGD_JSONRequestGetSize` e `GGD_JSONRequestGetFile` usam `GGD_SecureConnect_Read` para obter os dados JSON do soquete.

`GGD_JSONRequestStart`, `GGD_SecureConnect_Send` e `GGD_JSONRequestGetSize` devem ser chamados para receber os dados JSON da AWS IoT.

`GGD_GetIPandCertificateFromJSON`

Extrai o endereço IP e o certificado de núcleo do Greengrass a partir dos dados JSON. Você pode ativar a seleção automática definindo `xAutoSelectFlag` como `True`. A seleção automática encontra o primeiro dispositivo de núcleo ao qual seu dispositivo FreeRTOS pode se conectar. Para conectar-se a um núcleo do Greengrass, chame a função `GGD_SecureConnect_Connect`, transmitindo o endereço IP, a porta e o certificado do dispositivo de núcleo. Para usar a seleção manual, defina os seguintes campos do parâmetro `HostParameters_t`:

`pcGroupName`

O ID do grupo do Greengrass ao qual o núcleo pertence. Você pode usar o comando `aws greengrass list-groups` da CLI para encontrar o ID dos grupos do Greengrass.

`pcCoreAddress`

O ARN do núcleo do Greengrass ao qual você está se conectando.

Biblioteca coreHTTP

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Biblioteca cliente HTTP C para dispositivos de IoT pequenos (MCU ou MPU pequeno)

Introdução

A biblioteca coreHTTP é uma implementação cliente de um subconjunto do padrão HTTP/1.1. O padrão HTTP fornece um protocolo sem estado que é executado sobre o TCP/IP e é frequentemente usado em sistemas de informação de hipertexto distribuídos e colaborativos.

A biblioteca coreHTTP implementa um subconjunto do padrão de protocolo HTTP/1.1. Essa biblioteca foi otimizada para ocupar pouco espaço de memória. A biblioteca fornece uma API totalmente síncrona para que os aplicativos possam gerenciar completamente sua simultaneidade.

Ela usa somente buffers fixos, para que os aplicativos tenham controle total de sua estratégia de alocação de memória.

A biblioteca é escrita em C e criada para ser compatível com ISO C90 e MISRA C:2012. As únicas dependências da biblioteca são a biblioteca C padrão e a versão LTS (v12.19.1) do analisador http do Node.js. A biblioteca tem provas que mostram o uso seguro da memória e a ausência de alocação de heap e isso a torna adequada para microcontroladores do IoT, mas também totalmente portável para outras plataformas.

Ao usar conexões HTTP em aplicativos do IoT, recomendamos usar uma interface de transporte segura, como a usada pelo protocolo TLS, conforme demonstrado na Demonstração de autenticação mútua da coreHTTP.

Essa biblioteca pode ser usada gratuitamente e é distribuída sob a licença de código aberto do MIT.

Tamanho de código de coreHTTP (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -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
http.c (llhttp)	17,9	15,9
Estimativas totais	23,9 K	20,7 K

Biblioteca coreJSON

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

JSON (JavaScript Object Notation) é um formato de serialização de dados legível por humanos. Ele é amplamente usado para trocar dados, como com o serviço da solução Device Shadow da AWS IoT, e faz parte de muitas APIs, como a API REST do GitHub. O JSON é mantido como padrão pela Ecma International.

A biblioteca coreJSON fornece um analisador que é compatível com as pesquisas de chaves enquanto impõe exclusivamente a Sintaxe de intercâmbio de dados JSON ECMA-404 padrão. A biblioteca é escrita em C e projetada para estar em conformidade com ISO C90 e MISRA C:2012. Ela tem provas que mostram o uso seguro da memória e a ausência de alocação de heap e isso a torna adequada para microcontroladores do IoT, mas também totalmente portável para outras plataformas.

Uso de memória

A biblioteca coreJSON usa uma pilha interna para rastrear estruturas aninhadas em um documento JSON. A pilha existe durante o período de uma única chamada de função; ela não é preservada. O tamanho da pilha pode ser especificado definindo a macro `JSON_MAX_DEPTH`, cujo padrão é 32 níveis. Cada nível consome um único byte.

Tamanho de código de coreJSON (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
core_json.c	2,9 K	2,4 K
Estimativas totais	2,9 K	2,4 K

Biblioteca coreMQTT

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

A biblioteca coreMQTT é uma implementação de cliente do padrão MQTT (Message Queue Telemetry Transport). O padrão MQTT fornece um protocolo leve de mensagens de publicação/assinatura (ou PubSub) que é executado por meio de TCP/IP e frequentemente é usado em casos de uso de Máquina para máquina (M2M) e da Internet das Coisas (IoT).

A biblioteca coreMQTT é compatível com o padrão de protocolo MQTT 3.1.1. Essa biblioteca foi otimizada para ocupar pouco espaço de memória. O design dessa biblioteca abrange diferentes casos de uso, desde plataformas com recursos limitados usando somente mensagens MQTT PUBLISH de QoS 0 até plataformas repletas de recursos usando conexões MQTT PUBLISH de QoS 2 por meio do TLS (Transport Layer Security). A biblioteca fornece um menu de funções combináveis, que podem ser escolhidas e combinadas para atender exatamente às necessidades de um caso de uso específico.

A biblioteca é escrita em C e criada para ser compatível com ISO C90 e MISRA C:2012. Essa biblioteca MQTT não depende de bibliotecas adicionais, exceto das seguintes:

- A biblioteca C padrão
- Uma interface de transporte de rede implementada pelo cliente
- (Opcional) Uma função de horário da plataforma implementada pelo usuário

A biblioteca é desvinculada dos drivers de rede subjacentes por meio do fornecimento de uma especificação simples de interface de transporte de envio e recebimento. O autor do aplicativo pode selecionar uma interface de transporte existente ou implementar a própria interface, conforme adequado para o seu aplicativo.

A biblioteca fornece uma API de alto nível para conectar-se a um agente MQTT, assinar/cancelar a assinatura de um tópico, publicar uma mensagem em um tópico e receber mensagens de entrada. Essa API usa a interface de transporte descrita acima como um parâmetro e a usa para enviar e receber mensagens de e para o agente MQTT.

A biblioteca também expõe a API do serializador/desserializador de baixo nível. Essa API pode ser usada para criar um aplicativo do IoT simples que consiste apenas no subconjunto necessário da funcionalidade do MQTT, sem outras sobrecargas. A API do serializador/desserializador pode ser usada em conjunto com toda API de camada de transporte disponível, como soquetes, para enviar e receber mensagens de e para o agente.

Ao usar conexões MQTT em aplicativos do IoT, recomendamos usar uma interface de transporte segura, como a usada pelo protocolo TLS.

Essa biblioteca MQTT não tem dependências de plataforma, como threading ou sincronização. Essa biblioteca tem provas que demostram o uso seguro da memória e a ausência de alocação de heap, o que a torna adequada para microcontroladores do IoT, mas também totalmente portável para outras plataformas. Ela pode ser usada gratuitamente e é distribuída sob a licença de código aberto do MIT.

Tamanho de código de coreMQTT (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
core_mqtt.c	4,0 K	3,4 K
core_mqtt_state.c	1,7 K	1,3 K
core_mqtt_serializer.c	2,8 K	2,2 K
Estimativas totais	8,5 K	6,9 K

Biblioteca coreMQTT Agent

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

A biblioteca coreMQTT Agent é uma API de alto nível que adiciona segurança de threads ao Biblioteca coreMQTT. Ela permite criar uma tarefa de agente MQTT dedicada que gerencia uma conexão MQTT em segundo plano e sem precisar de nenhuma intervenção de outras tarefas. A biblioteca fornece equivalentes seguros de threads às APIs do coreMQTT, para que possa ser usada em ambientes com threads múltiplos.

O agente MQTT é uma tarefa independente (ou thread de execução). Ele obtém a segurança do thread ao ser a única tarefa que tem permissão para acessar a API da biblioteca MQTT. Ele serializa

o acesso isolando todas as chamadas da API MQTT em uma única tarefa e elimina a necessidade de semáforos ou outras primitivas de sincronização.

A biblioteca usa uma fila de mensagens segura para threads (ou outro mecanismo de comunicação entre processos) para serializar todas as solicitações para chamar as APIs do MQTT. A implementação do sistema de mensagens é desacoplada da biblioteca por meio de uma interface de mensagens, que permite que a biblioteca seja transferida para outros sistemas operacionais. A interface de mensagens é composta por funções para enviar e receber ponteiros para as estruturas de comando do agente e funções para alocar esses objetos de comando, o que permite que o escritor do aplicativo decida a estratégia de alocação de memória apropriada para seu aplicativo.

A biblioteca é escrita em C e criada para ser compatível com ISO C90 e MISRA C:2012. A biblioteca não depende de bibliotecas adicionais além da biblioteca Biblioteca coreMQTT e da C padrão. A biblioteca tem provas que mostram o uso seguro da memória e a ausência de alocação de heap, por isso ela pode ser usada para microcontroladores do IoT, mas também é totalmente portável para outras plataformas.

Essa biblioteca pode ser usada gratuitamente e é distribuída sob a licença de código aberto do MIT.

Tamanho de código da coreMQTT Agent (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -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
Estimativas totais	10,5 K	8,6 K

Biblioteca sem fios do AWS IoT

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

A biblioteca de atualização sem fios do AWS IoT permite gerenciar as notificações, o download e a verificação de atualizações de firmware para dispositivos do FreeRTOS usando HTTP ou MQTT como protocolo. Usando a biblioteca do agente OTA, você pode separar logicamente as atualizações de firmware e o aplicativo em execução nos dispositivos. O agente OTA pode compartilhar uma conexão de rede com o aplicativo. Ao compartilhar uma conexão de rede, você pode economizar uma quantidade significativa de RAM. Além disso, a biblioteca do agente OTA permite que você defina a lógica específica do aplicativo para testar, confirmar ou reverter uma atualização de firmware.

A Internet das Coisas (IoT) estende a conectividade com a Internet para dispositivos incorporados que tradicionalmente não estavam conectados. Estes dispositivos podem ser programados para comunicar dados utilizáveis pela Internet e podem ser monitorados e controlados remotamente. Com os avanços da tecnologia, esses dispositivos tradicionais incorporados estão obtendo recursos de Internet em espaços de consumo, industriais e corporativos em um ritmo acelerado.

Os dispositivos de IoT geralmente são implantados em grandes quantidades e geralmente em locais difíceis ou impraticáveis para um operador humano acessar. Imagine um cenário em que uma vulnerabilidade de segurança que pode expor dados seja descoberta. Nestes cenários, é importante atualizar os dispositivos afetados com correções de segurança de forma rápida e confiável. Sem a capacidade de realizar atualizações OTA, também pode ser difícil atualizar dispositivos que estão geograficamente dispersos. Um técnico que atualize esses dispositivos poderá custar caro, será demorado e, muitas vezes, será impraticável. O tempo necessário para atualizar esses dispositivos os deixa expostos a vulnerabilidades de segurança por um período mais longo. O recall desses dispositivos para atualização também será caro e poderá causar interrupções significativas aos consumidores devido ao tempo de inatividade.

As atualizações sem fios possibilitam a atualização do firmware do dispositivo sem um recall caro ou uma visita técnica. Este método adiciona os seguintes benefícios:

- Segurança: a capacidade de responder rapidamente às vulnerabilidades de segurança e aos bugs de software descobertos após a implantação dos dispositivos no campo.
- Inovação: os produtos podem ser atualizados com frequência à medida que novos atributos são desenvolvidos, impulsionando o ciclo de inovação. As atualizações podem entrar em vigor rapidamente com o mínimo de tempo de inatividade em comparação com os métodos tradicionais de atualização.
- Custo: as atualizações OTA podem reduzir significativamente os custos de manutenção em comparação com os métodos tradicionalmente usados para atualizar esses dispositivos.

Fornecer a funcionalidade do OTA requer as seguintes considerações de design:

- Comunicação segura: as atualizações devem usar canais de comunicação criptografados para evitar que os downloads sejam adulterados durante o trânsito.
- Recuperação: as atualizações podem falhar devido a fatores como conectividade de rede intermitente ou recebimento de uma atualização inválida. Nesses cenários, o dispositivo precisa conseguir retornar a um estado estável e evitar o bloqueio.
- Verificação de autor: as atualizações devem ser verificadas por serem de uma fonte confiável, junto com outras validações, como verificar a versão e a compatibilidade.

Para obter mais informações sobre como configurar atualizações OTA com o FreeRTOS consulte Atualizações sem fios do FreeRTOS.

Biblioteca sem fios do AWS IoT

A biblioteca OTA do AWS IoT permite gerenciar notificações de atualizações recém-disponíveis, baixá-las e realizar a verificação criptográfica das atualizações de firmware. Usando a biblioteca de cliente sem fios, você pode separar logicamente os mecanismos de atualização do firmware da aplicação em execução no seu dispositivo. A biblioteca cliente sem fios pode compartilhar uma conexão de rede com a aplicação, economizando memória em dispositivos com recursos limitados. Além disso, a biblioteca do cliente sem fios permite que você defina a lógica específica do aplicativo para testar, confirmar ou reverter uma atualização de firmware. A biblioteca oferece suporte a diferentes protocolos de aplicações, como o Message Queuing Telemetry Transport (MQTT) e o Hypertext Transfer Protocol (HTTP), e fornece várias opções de configuração que você pode ajustar de acordo com o tipo e as condições da sua rede.

As APIs desta biblioteca oferecem as seguintes funções principais:

- Registre-se para receber notificações ou pesquisar novas solicitações de atualização que estão disponíveis.
- Receba, analise e valide a solicitação de atualização.
- Baixe e verifique o arquivo de acordo com as informações na solicitação de atualização.
- Execute um autoteste antes de ativar a atualização recebida para garantir a validade funcional da atualização.
- Atualize o status do dispositivo.

Esta biblioteca usa serviços da AWS para gerenciar várias funções relacionadas à nuvem, como enviar atualizações de firmware, monitorar um grande número de dispositivos em várias regiões, reduzir o raio de explosão de implantações defeituosas e verificar a segurança das atualizações. Esta biblioteca pode ser usada com qualquer biblioteca MQTT ou HTTP.

As demonstrações dessa biblioteca demonstram atualizações remotas completas usando a biblioteca coreMQTT e os serviços da AWS em um dispositivo FreeRTOS.

Recursos

Esta é a interface completa do agente OTA:

`OTA_Init`

Inicializa o mecanismo OTA iniciando o atendente OTA ("Tarefa OTA") no sistema. Somente um atendente OTA pode existir.

`OTA_Shutdown`

Sinalize ao atendente OTA para desligar. Opcionalmente, o atendente OTA cancelará a assinatura de todos os tópicos de notificação de tarefas do MQTT, interromperá as tarefas do OTA em andamento, se houver, e limpará todos os recursos.

`OTA_GetState`

Obtém o estado atual do agente OTA.

`OTA_ActivateNewImage`

Ativa a mais nova imagem de firmware do microcontrolador recebida pelo OTA. (O status detalhado do trabalho agora deve ser teste automático.)

OTA_SetImageState

Define o estado de validação da imagem do firmware do microcontrolador atualmente em execução (teste, aceito ou rejeitado).

OTA_GetImageState

Obtém o estado da imagem do firmware do microcontrolador atualmente em execução (teste, aceito ou rejeitado).

OTA_CheckForUpdate

Solicita a próxima atualização OTA disponível do serviço de atualização OTA.

OTA_Suspend

Suspenda todas as operações do atendente OTA.

OTA_Resume

Reinicie as operações do atendente OTA.

OTA_SignalEvent

Sinalize um evento para a tarefa do atendente OTA.

OTA_EventProcessingTask

Loop de processamento de eventos do atendente OTA.

OTA_GetStatistics

Obtenha as estatísticas dos pacotes de mensagens OTA, que incluem o número de pacotes recebidos, enfileirados, processados e descartados.

OTA_Err_strerror

Código de erro na conversão de strings para erros OTA.

OTA_JobParse_strerror

Converta um código de erro do OTA Job Parsing em uma string.

OTA_PalStatus_strerror

Código de status para a conversão de strings para o status do sistema operacional PAL OTA.

OTA_OsStatus_strerror

Código de status para a conversão de strings para o status do sistema operacional OTA.

Referência de API

Para obter mais informações, consulte a Atualização do AWS IoT Over-the-Air: funções.

Exemplo de uso

Um aplicativo de dispositivo compatível com OTA típico usando o protocolo MQTT direciona o agente OTA usando a seguinte sequência de chamadas de API.

1. Conecte-se ao coreMQTT Agent do AWS IoT. Para obter mais informações, consulte Biblioteca coreMQTT Agent.
2. Inicialize o atendente OTA chamando `OTA_Init`, incluindo os buffers, as interfaces ota necessárias, o nome da coisa e o retorno de chamada da aplicação. O retorno de chamada implementa a lógica específica do aplicativo que é executada após a conclusão de um trabalho de atualização OTA.
3. Quando a atualização OTA estiver concluída, o FreeRTOS chamará o retorno de chamada da conclusão do trabalho com um dos seguintes eventos: `accepted`, `rejected`, ou `self test`.
4. Se a nova imagem de firmware tiver sido rejeitada (por exemplo, devido a um erro de validação), o aplicativo normalmente poderá ignorar a notificação e aguardar a próxima atualização.
5. Se a atualização for válida e tiver sido marcada como aceita, chame `OTA_ActivateNewImage` para redefinir o dispositivo e inicializar a nova imagem de firmware.

Portabilidade

Para obter informações sobre como fazer a portabilidade da funcionalidade OTA para sua plataforma, consulte Portabilidade da biblioteca OTA no Guia de portabilidade do FreeRTOS.

Uso de memória

Tamanho de código do OTA do AWS IoT (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -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

Tamanho de código do OTA do AWS IoT (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
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
Estimativas totais	12,5 K	11,3 K

Biblioteca corePKCS11

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Visão geral

O Padrão de criptografia de chave pública #11 define uma API independente de plataforma para gerenciar e usar tokens criptográficos. O PKCS #11 se refere à API definida pelo padrão e para o padrão. A API criptográfica de PKCS #11 abstrai armazenamento de chaves, propriedades de obtenção/definição para objetos criptográficos e semântica de sessão. É amplamente usada para manipular objetos criptográficos comuns, sendo importante porque as funções que ela especifica permitem que o software do aplicativo use, crie, modifique e exclua objetos criptográficos, sem expor esses objetos à memória do aplicativo. Por exemplo, as integrações de referência do FreeRTOS da AWS usam um pequeno subconjunto da API PKCS #11 para acessar a chave secreta (privada) necessária para criar uma conexão de rede autenticada e protegida pelo protocolo Transport Layer Security (TLS) sem que o aplicativo "veja" a chave.

A biblioteca corePKCS11 contém uma implementação simulada baseada em software da interface PKCS #11 (API) que usa a funcionalidade criptográfica fornecida pelo TLS Mbed. O uso de uma simulação de software permite rápido desenvolvimento e flexibilidade, mas o esperado é que você substitua a simulação por uma implementação específica para o armazenamento seguro de chaves usado em nos dispositivos de produção. Geralmente, fornecedores de criptoprocessadores

seguros, como Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element ou outros tipos de enclaves de hardware seguros, distribuem uma implementação de PKCS #11 com o hardware. O objetivo da biblioteca simulada exclusiva do software corePKCS11 é, portanto, fornecer uma implementação PKCS #11 não específica de hardware que permita prototipagem e desenvolvimento rápidos antes de alternar para uma implementação PKCS #11 específica do criptoprocessador em dispositivos de produção.

Somente um subconjunto do padrão PKCS #11 é implementado, com foco em operações envolvendo chaves assimétricas, geração de números aleatórios e hashes. Os casos de uso alvo incluem gerenciamento de certificados e chaves para autenticação TLS e verificação de assinatura de código em dispositivos pequenos incorporados. Consulte o arquivo `pkcs11.h` (obtido do OASIS, o corpo padrão) no repositório de código-fonte do FreeRTOS. Na implementação de referência do FreeRTOS, as chamadas de API do PKCS#11 são feitas pela interface auxiliar do TLS para executar a autenticação do cliente TLS durante `SOCKETS_Connect`. Chamadas de API do PKCS#11 também são feitas pelo nosso fluxo de trabalho de provisionamento de desenvolvedor único para importar um certificado de cliente TLS e uma chave privada para autenticação do agente MQTT da AWS IoT. Esses dois casos de uso, o provisionamento e a autenticação de cliente TLS, exigem a implementação de apenas um pequeno subconjunto do padrão de interface PKCS#11.

Recursos

O subconjunto do PKCS#11 a seguir é usado. Essa lista é aproximadamente a ordem em que as rotinas são chamadas no suporte ao provisionamento, à autenticação de cliente TLS e à limpeza. Para descrições detalhadas das funções, consulte a documentação do PKCS#11 fornecida pelo comitê padrão.

API de configuração geral e desativação

- `C_Initialize`
- `C_Finalize`
- `C_GetFunctionList`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_OpenSession`
- `C_CloseSession`
- `C_Login`

API de provisionamento

- `C_CreateObject CKO_PRIVATE_KEY` (para a chave privada do dispositivo)
- `C_CreateObject CKO_CERTIFICATE` (para o certificado do dispositivo e o certificado de verificação de código)
- `C_GenerateKeyPair`
- `C_DestroyObject`

Autenticação de 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`

Suporte ao sistema criptográfico assimétrico

A implementação de referência do FreeRTOS usa PKCS #11 RSA de 2048 bits (somente assinatura) e ECDSA com as curvas do NIST P-256. As instruções a seguir descrevem como criar uma coisa da AWS IoT baseada em um certificado de cliente P-256.

Certifique-se de estar usando as seguintes versões (ou mais recentes) da AWS CLI e do 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
```

O procedimento a seguir considera que você usou o comando `aws configure` para configurar o AWS CLI. Para obter mais informações, consulte Configuração rápida com o `aws configure` no Manual do usuário do AWS Command Line Interface.

Como criar uma coisa da AWS IoT baseada em um certificado de cliente P-256

1. Crie uma coisa do AWS IoT.

   ```
   aws iot create-thing --thing-name thing-name
   ```

2. Use o OpenSSL para criar uma chave 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. Crie uma solicitação de inscrição de certificado assinada pela chave criada na etapa 2.

   ```
   openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
   ```

4. Envie a solicitação de inscrição de certificado ao 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. Anexe o certificado (referenciado pela saída do ARN pelo comando anterior) à coisa.

   ```
   aws iot attach-thing-principal --thing-name thing-name \
     --principal "arn:aws:iot:us-
   east-1:123456789012:cert/
   86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
   ```

6. Crie uma política. (Essa política é muito permissiva. Ela deve ser usada somente para fins de desenvolvimento.)

   ```
   aws iot create-policy --policy-name FullControl --policy-document file://
   policy.json
   ```

A seguir, uma listagem do arquivo policy.json especificado no comando `create-policy`. Você pode omitir a ação `greengrass:*` se não quiser executar a demonstração do FreeRTOS para a conectividade e a descoberta do Greengrass.

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

7. Anexe o principal (certificado) e a política à coisa.

```
aws iot attach-principal-policy --policy-name FullControl \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

Agora, siga as etapas na seção Conceitos básicos do AWS IoT desse guia. Não se esqueça de copiar o certificado e a chave privada que você criou no arquivo `aws_clientcredential_keys.h`. Copie o nome da coisa em `aws_clientcredential.h`.

> **Note**
>
> O certificado e a chave privada são codificados para fins de demonstração somente. Por este motivo, os aplicativos devem armazenar esses arquivos em um local seguro.

Portabilidade

Para obter informações sobre a portabilidade da biblioteca corePKCS11 para sua plataforma, consulte Portabilidade da biblioteca corePKCS11 no Guia de portabilidade do FreeRTOS.

Uso de memória

Tamanho de código da corePKCS11 (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -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
Estimativas totais	10,2 K	8,6 K

Biblioteca de Secure Sockets

> **Important**
>
> Essa biblioteca está hospedada no RTOS repositório Amazon-Free, que está obsoleto. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um RTOS projeto gratuito existente com base no repositório Amazon-FreeRTOS, agora obsoleto, consulte o. Guia de migração do repositório Github do Amazon FreeRTOS

Visão geral

Você pode usar a biblioteca Free RTOS Secure Sockets para criar aplicativos incorporados que se comunicam com segurança. A biblioteca foi projetada para facilitar a integração de desenvolvedores de software de várias bases de programação de rede.

A biblioteca Free RTOS Secure Sockets é baseada na interface de soquetes Berkeley, com uma opção adicional de comunicação segura por protocolo. TLS Para obter informações sobre as diferenças entre a biblioteca Free RTOS Secure Sockets e a interface de soquetes Berkeley, consulte a Secure Sockets `SOCKETS_SetSockOpt` Reference. API

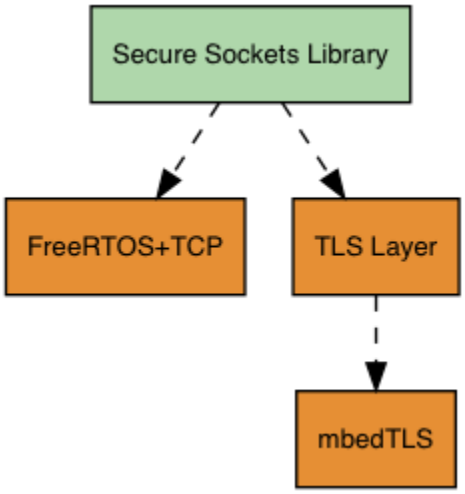
> **Note**
>
> Atualmente, somente o clienteAPIs, além de uma implementação de IP leve (LWIP) do `Bind` API lado do servidor, é compatível com Free Secure Sockets. RTOS

Dependências e requisitos

A biblioteca Free RTOS Secure Sockets depende de uma pilha TCP /IP e de uma implementação. TLS O Ports for Free RTOS atende a essas dependências de uma das três maneiras:

- Uma implementação personalizada de TCP /IP e TLS
- Uma implementação personalizada de TCP /IP e da RTOS TLS camada livre com mbed TLS
- Free RTOS + TCP e a RTOS TLS camada Free com incorporação TLS

O diagrama de dependências abaixo mostra a implementação de referência incluída na biblioteca Free RTOS Secure Sockets. Essa implementação de referência suporta TLS TCP e/IP sobre Ethernet e Wi-Fi com Free RTOS + TCP e mbed TLS como dependências. Para obter mais informações sobre a RTOS TLS camada Livre, consulteTransport Layer Security.



Recursos

Os recursos gratuitos da biblioteca RTOS Secure Sockets incluem:

- Uma interface padrão baseada em soquetes Berkeley
- Thread-safe APIs para enviar e receber dados
- E asy-to-enable TLS

Solução de problemas

Códigos de erro

Os códigos de erro que a biblioteca Free RTOS Secure Sockets retorna são valores negativos. Para obter mais informações sobre cada código de erro, consulte Códigos de erro do Secure Sockets na Referência do Secure Sockets API.

> **Note**
>
> Se o Free RTOS Secure Sockets API retornar um código de erro, oBiblioteca coreMQTT, que depende da biblioteca Free RTOS Secure Sockets, retornará o código de erro. `AWS_IOT_MQTT_SEND_ERROR`

Suporte ao desenvolvedor

A biblioteca Free RTOS Secure Sockets inclui duas macros auxiliares para lidar com endereços IP:

`SOCKETS_inet_addr_quick`

Essa macro converte um endereço IP expresso por quatro octetos numéricos separados em um endereço IP expresso como um número de 32 bits em ordem de bytes de rede.

`SOCKETS_inet_ntoa`

Essa macro converte um endereço IP expresso como um número de 32 bits em ordem de bytes de rede em uma string com notação de ponto decimal.

Restrições de uso

Somente TCP soquetes são suportados pela biblioteca Free RTOS Secure Sockets. UDPsoquetes não são suportados.

APIsOs servidores não são suportados pela biblioteca Free RTOS Secure Sockets, exceto por uma implementação de IP leve (LWIP) do lado do servidor. `Bind` API O cliente APIs é suportado.

Inicialização

Para usar a biblioteca Free RTOS Secure Sockets, você precisa inicializar a biblioteca e suas dependências. Para inicializar a biblioteca de Secure Sockets, use o seguinte código na aplicação:

```
BaseType_t xResult = pdPASS;
xResult = SOCKETS_Init();
```

As bibliotecas dependentes devem ser inicializadas separadamente. Por exemplo, se Free RTOS + TCP for uma dependência, você também precisará invocar `FreeRTOS_IPInit`em seu aplicativo.

APIreferência

Para obter uma API referência completa, consulte Secure Sockets API Reference.

Exemplo de uso

O código a seguir conecta um cliente a um 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 obter um exemplo completo, consulte Demonstração do cliente Echo de Secure Sockets.

Portabilidade

O Free RTOS Secure Sockets depende de uma pilha TCP /IP e de uma implementação. TLS Dependendo da pilha, para transferir a biblioteca de Secure Sockets, será necessário transferir alguns dos seguintes:

- A pilha Free RTOS TCP TCP +/IP
- A Biblioteca corePKCS11
- A Transport Layer Security

Para obter mais informações sobre portabilidade, consulte Porting the Secure Sockets Library no Free RTOS Porting Guide.

Biblioteca de Device Shadow da AWS IoT

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

Você pode usar a biblioteca do AWS IoT Device Shadow para armazenar e recuperar o estado atual (a sombra) de cada dispositivo registrado. A sombra do dispositivo é uma representação virtual persistente do seu dispositivo com a qual você pode interagir em seus aplicativos da web mesmo se o dispositivo estiver offline. O estado do dispositivo é capturado assim como o da sua sombra em um documento JSON. Você pode enviar comandos para o serviço AWS IoT Device Shadow por meio de MQTT ou HTTP para consultar o estado mais recente conhecido do dispositivo ou para alterar o estado. A sombra de cada dispositivo é identificada exclusivamente pelo nome da coisa correspondente, pela representação de um dispositivo específico ou entidade lógica na nuvem da AWS. Para obter mais informações, consulte Gerenciamento de dispositivos no AWS IoT. Mais detalhes sobre sombras podem ser encontrados na Documentação do AWS IoT.

A biblioteca do AWS IoT Device Shadow não depende de bibliotecas adicionais além da biblioteca C padrão. Ela também não depende de plataformas, como threading ou sincronização. Ela pode ser usada com toda biblioteca MQTT e toda biblioteca JSON.

Essa biblioteca pode ser usada gratuitamente e é distribuída sob a licença de código aberto do MIT.

Tamanho de código do AWS IoT Device Shadow (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
shadow.c	1,2 K	0,9 K
Estimativas totais	1,2 K	0,9 K

Biblioteca Trabalhos do AWS IoT

> **Note**
>
> O conteúdo desta página pode não estar atualizado. Consulte a página da biblioteca do FreeRTOS.org para obter a atualização mais recente.

Introdução

Trabalhos do AWS IoT é um serviço que notifica um ou mais dispositivos conectados de um trabalho pendente. Você pode usar um trabalho para gerenciar sua frota de dispositivos, atualizar o firmware

e os certificados de segurança nos dispositivos ou realizar tarefas administrativas, como reiniciar dispositivos e realizar diagnósticos. Para obter mais informações, consulte Trabalhos no Guia do desenvolvedor do AWS IoT. As interações com o serviço Trabalhos do AWS IoT usam o MQTT, um protocolo leve de publicação e assinatura. Essa biblioteca fornece uma API para compor e reconhecer as strings de tópicos do MQTT usadas pelo serviço Trabalhos do AWS IoT.

A biblioteca Trabalhos do AWS IoT é escrita em C e criada para ser compatível com ISO C90 e MISRA C:2012. A biblioteca não depende de bibliotecas adicionais além da biblioteca C padrão. Ela pode ser usada com toda biblioteca MQTT e toda biblioteca JSON. A biblioteca tem provas que mostram o uso seguro da memória e a ausência de alocação de heap e isso a torna adequada para microcontroladores do IoT, mas também totalmente portável para outras plataformas.

Essa biblioteca pode ser usada gratuitamente e é distribuída sob a licença de código aberto do MIT.

Tamanho de código de Trabalhos do AWS IoT (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
jobs.c	1,9 K	1,6 K
Estimativas totais	1,9 K	1,6 K

Transport Layer Security

> **Important**
>
> Essa biblioteca está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

A interface de Transport Layer Security (TLS) do FreeRTOS é um wrapper fino e opcional usado para abstrair detalhes de implementação criptográfica da interface Secure Sockets Layer (SSL) acima dela na pilha de protocolos. A finalidade da interface TLS é tornar a biblioteca de criptografia de software atual, mbed TLS, fácil de substituir por uma implementação alternativa para negociação de protocolo TLS e primitivos de criptografia. A interface TLS pode ser trocada sem quaisquer

alterações necessárias na interface SSL. Consulte `iot_tls.h` no repositório de código-fonte do FreeRTOS.

A interface TLS é opcional porque você pode escolher a interface diretamente do SSL em uma biblioteca de criptografia. A interface não é usada para soluções de MCU que incluem uma implementação de descarregamento de pilha completa de TLS e transporte de rede.

Para obter mais informações sobre portabilidade da interface TLS, consulte Portabilidade da biblioteca TLS no Guia de portabilidade do FreeRTOS.

Biblioteca de Wi-Fi

> **Important**
>
> Essa biblioteca está hospedada no RTOS repositório Amazon-Free, que está obsoleto. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um RTOS projeto gratuito existente com base no repositório Amazon-FreeRTOS, agora obsoleto, consulte o. Guia de migração do repositório Github do Amazon FreeRTOS

Visão geral

A biblioteca RTOS Wi-Fi gratuita resume as implementações de Wi-Fi específicas de portas em uma linguagem comum API que simplifica o desenvolvimento e a portabilidade de aplicativos para todas as placas qualificadas gratuitamente com recursos RTOS de Wi-Fi. Usando esse comumAPI, os aplicativos podem se comunicar com sua pilha sem fio de nível inferior por meio de uma interface comum.

Dependências e requisitos

A biblioteca RTOS Wi-Fi gratuita requer o TCP núcleo Free RTOS +.

Recursos

A biblioteca Wi-Fi inclui os seguintes recursos:

- Support forWEP,WPA,WPA2, and WPA3 authentication
- Verificação do ponto de acesso
- Gerenciamento de energia
- Criação de perfis na rede

Para obter mais informações sobre os recursos da biblioteca Wi-Fi, veja abaixo.

Modos Wi-Fi

Os dispositivos Wi-Fi podem estar em um dos três modos: Estação, Ponto de acesso ou P2P. Você pode obter o modo atual de um dispositivo Wi-Fi chamando `WIFI_GetMode`. Defina o modo wi-fi de um dispositivo chamando `WIFI_SetMode`. Alternar modos chamando `WIFI_SetMode` desconecta o dispositivo, se já estiver conectado a uma rede.

Modo Estação

Defina o dispositivo para o modo Estação a fim de conectar a placa a um ponto de acesso existente.

Modo Ponto de acesso (AP)

Defina o dispositivo para o modo AP para tornar o dispositivo um ponto de acesso para outros dispositivos se conectarem. Quando seu dispositivo está no modo AP, você pode conectar outro dispositivo ao seu RTOS dispositivo gratuito e configurar as novas credenciais de Wi-Fi. Para configurar o modo de AP, chame `WIFI_ConfigureAP`. Para colocar o dispositivo no modo AP, chame `WIFI_StartAP`. Para desativar o modo AP, chame `WIFI_StopAP`.

> **Note**
>
> RTOSBibliotecas gratuitas não fornecem provisionamento Wi-Fi no modo AP. Você deve fornecer a funcionalidade adicional, incluindo DHCP os recursos HTTP do servidor, para obter suporte total ao modo AP.

Modo P2P

Defina o dispositivo para o modo P2P a fim de permitir que vários dispositivos se conectem uns aos outros diretamente, sem um ponto de acesso.

Segurança

O Wi-Fi API suportaWEP,WPA,WPA2, e tipos WPA3 de segurança. Quando um dispositivo está no modo Estação, é necessário especificar o tipo de segurança de rede ao chamar a função `WIFI_ConnectAP`. Quando um dispositivo está no modo AP, ele pode ser configurado para usar qualquer um dos tipos de segurança compatíveis:

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

Verificação e conexão

Para procurar pontos de acesso próximos, defina o dispositivo para o modo Estação e chame a função `WIFI_Scan`. Se você encontrar uma rede desejada durante a verificação, poderá se conectar à rede chamando `WIFI_ConnectAP` e fornecendo as credenciais da rede. Você pode desconectar um dispositivo Wi-Fi da rede chamando `WIFI_Disconnect`. Para obter mais informações sobre a verificação e a conexão, consulte Exemplo de uso e APIreferência.

Gerenciamento de energia

Diferentes dispositivos Wi-Fi possuem diferentes requisitos de energia, dependendo da aplicação e das fontes de energia disponíveis. Um dispositivo pode ficar sempre ligado para reduzir a latência ou pode ficar intermitentemente conectado e alternar para um modo de baixo consumo quando o Wi-Fi não for necessário. A interface API suporta vários modos de gerenciamento de energia, como sempre ligado, baixo consumo de energia e modo normal. Você define o modo de energia para um dispositivo usando a função `WIFI_SetPMMode`. Você pode obter o modo de energia atual de um dispositivo chamando a função `WIFI_GetPMMode`.

Perfis de rede

A biblioteca Wi-Fi permite salvar perfis de rede na memória não volátil dos dispositivos. Isso permite que você salve as configurações de rede para que elas possam ser recuperadas quando um dispositivo se reconectar a uma rede Wi-Fi, eliminando a necessidade de provisionar os dispositivos novamente depois de terem sido conectados a uma rede. `WIFI_NetworkAdd` adiciona um perfil de rede. `WIFI_NetworkGet` recupera um perfil de rede. `WIFI_NetworkDel` exclui um perfil de rede. O número de perfis que você pode salvar depende da plataforma.

Configuração

Para usar a biblioteca Wi-Fi, é necessário definir vários identificadores em um arquivo de configuração. Para obter informações sobre esses identificadores, consulte a APIreferência.

> **Note**
>
> A biblioteca não inclui o arquivo de configuração necessário. Você deve criar um. Ao criar o arquivo de configuração, lembre-se de incluir os identificadores de configuração específicos exigidos pela placa.

Inicialização

Antes de usar a biblioteca Wi-Fi, você precisa inicializar alguns componentes específicos da placa, além dos componentes gratuitos. RTOS Ao usar o arquivo `vendors/vendor/boards/board/aws_demos/application_code/main.c` como um modelo para a inicialização, faça o seguinte:

1. Remova o exemplo de lógica de conexão Wi-Fi em `main.c` caso a aplicação manuseie conexões Wi-Fi. Substitua a seguinte chamada da função `DEMO_RUNNER_RunDemos()`:

   ```
   if( SYSTEM_Init() == pdPASS )
       {
       ...
           DEMO_RUNNER_RunDemos();
       ...
           }
   ```

   Por uma chamada a sua própria aplicação:

   ```
   if( SYSTEM_Init() == pdPASS )
       {
       ...
           // This function should create any tasks
           // that your application requires to run.
           YOUR_APP_FUNCTION();
       ...
           }
   ```

2. Chame `WIFI_On()` para inicializar e ativar o chip Wi-Fi.

   > **Note**
   >
   > Algumas placas podem exigir a inicialização de hardware adicional.

3. Passe uma estrutura `WIFINetworkParams_t` configurada para `WIFI_ConnectAP()` a fim de conectar sua placa a uma rede Wi-Fi disponível. Para obter mais informações sobre a estrutura `WIFINetworkParams_t`, consulte Exemplo de uso e APIreferência.

APIreferência

Para obter uma API referência completa, consulte APIReferência de Wi-Fi.

Exemplo de uso

Conexão a um AP conhecido

```
#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
}
```

Escaneando por áreas próximas APs

```
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);
}
```

Portabilidade

A implementação de `iot_wifi.c` precisa implementar as funções definidas em `iot_wifi.h`. No mínimo, a implementação precisa retornar `eWiFiNotSupported` para qualquer função não essencial ou incompatível.

Para obter mais informações sobre como portar a biblioteca Wi-Fi, consulte Portando a biblioteca Wi-Fi no Free RTOS Porting Guide.

Demonstrações do FreeRTOS

O FreeRTOS inclui alguns aplicativos de demonstração na pasta `demos`, no diretório principal do FreeRTOS. Todos os exemplos que podem ser executados pelo FreeRTOS são exibidos na pasta `common`, em `demos`. Há também uma pasta para cada plataforma qualificada para FreeRTOS na pasta `demos`.

Antes de testar os aplicativos de demonstração, recomendamos que você conclua o tutorial em Conceitos básicos do FreeRTOS. Ele mostra como configurar e executar a demonstração da coreMQTT Agent.

Execução da demonstração do FreeRTOS

Os tópicos a seguir mostram como configurar e executar as demonstrações do FreeRTOS:

- Aplicações de demonstração do Bluetooth Low Energy
- Bootloader de demonstração para o Microchip Curiosity PIC32MZEF

- AWS IoT Device Defender demonstração
- Aplicativo de demonstração do AWS IoT Greengrass Discovery V1
- AWS IoT Greengrass V2
- Demonstrações de coreHTTP
- Demonstração da biblioteca Trabalhos do AWS IoT
- Demonstrações do coreMQTT
- Aplicativo de demonstração de atualizações remotas
- Demonstração do cliente Echo de Secure Sockets
- Aplicativo de demonstração do Device Shadow da AWS IoT

A função `DEMO_RUNNER_RunDemos`, localizada no arquivo *`freertos`*`/demos/demo_runner/iot_demo_runner.c`, inicializa um thread desanexado no qual um único aplicativo de demonstração é executado. Por padrão, `DEMO_RUNNER_RunDemos` chama e inicia somente a demonstração da coreMQTT Agent. Dependendo da configuração que você selecionou quando fez download do FreeRTOS, e do local que você fez o download dele, as outras funções do executor de exemplo podem iniciar por padrão. Para habilitar um aplicativo de demonstração, abra o arquivo *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h` e defina a demonstração a ser executada.

> **Note**
>
> Nem todas as combinações de exemplos funcionam em conjunto. Dependendo da combinação, o software pode não ser executado no destino selecionado devido a restrições de memória. Recomendamos que você execute uma demonstração por vez.

Configuração das demonstrações

As demonstrações foram configuradas para que você comece rapidamente. Você pode alterar algumas das configurações do projeto para criar uma versão que é executada em sua plataforma. É possível encontrar os arquivos de configuração em `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files`.

Aplicações de demonstração do Bluetooth Low Energy

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Visão geral

A Bluetooth Low Energy do FreeRTOS inclui três aplicações de demonstração:

- Demonstração do MQTT por Bluetooth Low Energy

  Este aplicação demonstra como usar o MQTT pelo serviço Bluetooth Low Energy.
- Demonstração do Provisionamento de Wi-Fi

  Esta aplicação demonstra como usar o serviço de provisionamento de Wi-Fi Bluetooth Low Energy.
- Demonstração do Servidor de atributos genéricos

  Esta aplicação demonstra como usar as APIs do middleware Bluetooth Low Energy do FreeRTOS para criar um servidor GATT simples.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Pré-requisitos

Para acompanhar essas demonstrações, você precisa de um microcontrolador com recursos do Bluetooth Low Energy. Você também precisa do SDK do iOS para dispositivos Bluetooth do FreeRTOS ou do SDK do Android para dispositivos Bluetooth do FreeRTOS.

Configure o AWS IoT Amazon Cognito para Freertos Bluetooth Low Energy

Para conectar seus AWS IoT dispositivos ao MQTT, você precisa configurar o Amazon AWS IoT Cognito.

Para configurar AWS IoT

1. Configure uma AWS conta em https://aws.amazon.com/.
2. Abra o console AWS IoT e, no painel de navegação, escolha Manage (Gerenciar) e depois Things (Coisas).
3. Escolha Create (Criar) e Create a single thing (Criar uma única coisa).
4. Insira um nome para o dispositivo e escolha Next (Próximo).
5. Se você estiver conectando seu microcontrolador à nuvem por meio de um dispositivo móvel, escolha Create thing without certificate (Criar coisa sem certificado). Como os SDKs móveis usam o Amazon Cognito para a autenticação de dispositivos, não é necessário criar um certificado de dispositivo para demonstrações que usam Bluetooth Low Energy.

   Se você estiver conectando seu microcontrolador à nuvem diretamente por Wi-Fi, escolha Create certificate (Criar certificado), Activate (Ativar) e faça download do certificado da coisa, da chave pública e da chave privada.
6. Escolha a coisa que você acabou de criar na lista de coisas registradas e escolha Interact (Interagir) na página da coisa. Anote o endpoint da API AWS IoT REST.

Para obter mais informações sobre a configuração, consulte Introdução ao AWS IoT.

Como criar um grupo de usuários do Amazon Cognito

1. Abra o console do Amazon Cognito e escolha Gerenciar grupos de usuários.
2. Selecione Criar um grupo de usuários.
3. Atribua um nome ao grupo de usuários e escolha Review defaults (Revisar padrões).
4. No painel de navegação, escolha App clients (Clientes de aplicação) e Add an app client (Adicionar um cliente de aplicação).
5. Insira um nome para o cliente de aplicação e escolha Create app client (Criar cliente de aplicação).
6. No painel de navegação, escolha Review (Revisar) e, depois, Create pool (Criar grupo).

Anote o ID do grupo que aparecer na página General Settings (Configurações gerais) do grupo de usuários.

7. No painel de navegação, escolha App clients (Clientes de aplicação) e Show details (Mostrar detalhes). Anote o ID e o segredo do cliente de aplicação.

Como criar um banco de identidades do Amazon Cognito

1. Abra o console do Amazon Cognito e escolha Gerenciar bancos de identidades.
2. Insira um nome para o grupo de identidades.
3. Expanda Authentication providers (Provedores de autenticação), escolha a guia Cognito e insira os IDs do grupo de usuários e do cliente de aplicação.
4. Selecione Criar grupo.
5. Expanda View Details (Exibir detalhes) e anote os dois nomes de função do IAM. Escolha Permitir e crie os perfis do IAM para identidades autenticadas e não autenticadas acessarem ao Amazon Cognito.
6. Escolha Edit identity pool (Editar grupo de identidades). Anote o ID do grupo de identidades. Ele deve ter o formato `us-west-2:12345678-1234-1234-1234-123456789012`.

Para obter mais informações sobre os conceitos básicos do Amazon Cognito, consulte Conceitos básicos do Amazon Cognito.

Como criar e anexar uma política do IAM à identidade autenticada

1. Abra o console do IAM e, no painel de navegação, escolha Perfis.
2. Encontre e escolha a função da sua identidade autenticada, escolha Attach policies (Anexar políticas) e, em seguida, escolha Add inline policy (Adicionar política em linha).
3. Escolha a guia JSON e cole o JSON a seguir:

```
{
   "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. Escolha Review policy (Revisar política), insira um nome para a política e escolha Create policy (Criar política).

Mantenha suas informações AWS IoT e do Amazon Cognito em mãos. Você precisa do endpoint e dos IDs para autenticar seu aplicativo móvel na AWS nuvem.

Configuração do ambiente do FreeRTOS para Bluetooth Low Energy

Para configurar o ambiente, é necessário fazer download do FreeRTOS com a Biblioteca de Bluetooth Low Energy no microcontrolador, além de fazer download do SDK móvel e configurá-lo para dispositivos Bluetooth do FreeRTOS em seu dispositivo móvel.

Como configurar o ambiente do microcontrolador com o Bluetooth Low Energy do FreeRTOS

1. Baixe ou clone FreeRTOS de. GitHub Consulte o arquivo README.md para obter instruções.
2. Configure o FreeRTOS no seu microcontrolador.

   Para obter informações sobre os conceitos básicos do FreeRTOS em um microcontrolador qualificado para o FreeRTOS, consulte o guia da sua placa em Conceitos básicos do FreeRTOS.

   > **Note**
   >
   > Você pode executar as demonstrações em todo microcontrolador habilitado para o Bluetooth Low Energy com o FreeRTOS e as bibliotecas Bluetooth Low Energy do FreeRTOS obtidas por portabilidade. Atualmente, o projeto de demonstração MQTT

por Bluetooth Low Energy do FreeRTOS foi totalmente migrado para os seguintes dispositivos habilitados para Bluetooth Low Energy:

- Espressif ESP32- DevKit C e o ESP-WROVER-KIT
- Nordic nRF52840-DK

Componentes comuns

As aplicações de demonstração do FreeRTOS têm dois componentes comuns:

- Gerenciador de rede
- aplicação de demonstração do SDK móvel de Bluetooth Low Energy

Gerenciador de rede

O Gerenciador de rede gerencia a conexão de rede do microcontrolador. Ele está localizado no diretório do FreeRTOS, em `demos/network_manager/aws_iot_network_manager.c`. Se o gerenciador de rede está habilitado para Wi-Fi e Bluetooth Low Energy, as demonstrações começam com Bluetooth Low Energy por padrão. Se a conexão Bluetooth Low Energy for interrompida e sua placa for habilitada para Wi-Fi, o Gerenciador de rede alternará para uma conexão Wi-Fi disponível para evitar que a rede seja desconectada.

Para habilitar um tipo de conexão de rede com o Gerenciador de rede, adicione o tipo de conexão de rede ao parâmetro `configENABLED_NETWORKS` em `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h` (em que *`vendor`* é o nome do fornecedor e *`board`* é o nome da placa que está sendo usada para executar as demonstrações).

Por exemplo, se você estiver com Bluetooth Low Energy e rede Wi-Fi ativadas, a linha que começa com `#define configENABLED_NETWORKS` em `aws_iot_network_config.h` mostrará:

```
#define  configENABLED_NETWORKS  ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

Para obter uma lista dos tipos de conexão de rede compatíveis no momento, consulte as linhas que começam com `#define AWSIOT_NETWORK_TYPE` em `aws_iot_network.h`.

aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS

O aplicativo de demonstração do FreeRTOS Bluetooth Low Energy Mobile SDK está localizado GitHub no SDK do Android para dispositivos Bluetooth do FreeRTOS, abaixo, e no SDK do iOS para dispositivos Bluetooth `amazon-freertos-ble-android-sdk/app` do FreeRTOS, abaixo. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo` Neste exemplo, usamos capturas de tela da versão para iOS da aplicação móvel de demonstração.

> **Note**
>
> Se você estiver usando um dispositivo iOS, precisará do Xcode para criar a aplicação móvel de demonstração. Se você estiver usando um dispositivo Android, poderá usar o Android Studio para criar a aplicação móvel de demonstração.

Para configurar a aplicação de demonstração do SDK para iOS

Ao definir variáveis de configuração, use o formato dos valores de espaços reservados fornecidos nos arquivos de configuração.

1. Confirme se o SDK do iOS para dispositivos Bluetooth do FreeRTOS está instalado.
2. Emita o comando a seguir de `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/`:

   ```
   $ pod install
   ```

3. Abra o projeto `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace` com Xcode e altere a conta do desenvolvedor assinante para a sua conta.
4. Crie uma AWS IoT política na sua região (caso ainda não tenha feito isso).

   > **Note**
   >
   > Essa política é diferente da política do IAM criada para a identidade autenticada do Amazon Cognito.

   a. Abra o console de AWS IoT.

b. No painel de navegação, escolha Secure (Seguro), Policies (Políticas) e Create (Criar). Insira um nome para identificar a política. Na seção Add statements (Adicionar instruções), escolha Advanced mode (Modo avançado). Copie e cole o seguinte JSON na janela do editor de política. Substitua *aws-region* e aws-account por sua região *e* ID da conta. 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. Escolha Criar.

5. Abra `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift` e redefina as variáveis a seguir:

   - `region`: Sua AWS região.
   - `iotPolicyName`: O nome AWS IoT da sua apólice.
   - `mqttCustomTopic`: o tópico MQTT no qual você deseja publicar.

6. Abra o `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json`.

Em `CognitoIdentity`, redefina as variáveis a seguir:

- `PoolId`: ID de banco de identidades do Amazon Cognito.
- `Region`: Sua AWS região.

Em `CognitoUserPool`, redefina as variáveis a seguir:

- `PoolId`: ID de grupo de usuários do Amazon Cognito.
- `AppClientId`: o ID do cliente da aplicação.
- `AppClientSecret`: o segredo do cliente da aplicação.
- `Region`: Sua AWS região.

Para configurar a aplicação de demonstração do SDK para Android

Ao definir variáveis de configuração, use o formato dos valores de espaços reservados fornecidos nos arquivos de configuração.

1. Confirme se o SDK do Android para dispositivos Bluetooth do FreeRTOS está instalado.
2. Crie uma AWS IoT política na sua região (caso ainda não tenha feito isso).

   > **Note**
   >
   > Essa política é diferente da política do IAM criada para a identidade autenticada do Amazon Cognito.

   a. Abra o console de AWS IoT.

   b. No painel de navegação, escolha Secure (Seguro), Policies (Políticas) e Create (Criar). Insira um nome para identificar a política. Na seção Add statements (Adicionar instruções), escolha Advanced mode (Modo avançado). Copie e cole o seguinte JSON na janela do editor de política. Substitua *`aws-region`* e aws-account por sua região *`e`* ID da conta. 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. Escolha Criar.

3. Abra https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/ DemoConstants .java e redefina as seguintes variáveis:

   - `AWS_IOT_POLICY_NAME`: O nome AWS IoT da sua apólice.
   - `AWS_IOT_REGION`: Sua AWS região.

4. Abra https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/res/raw/awsconfiguration.json.

   Em `CognitoIdentity`, redefina as variáveis a seguir:

   - `PoolId`: ID de banco de identidades do Amazon Cognito.
   - `Region`: Sua AWS região.

   Em `CognitoUserPool`, redefina as variáveis a seguir:

   - `PoolId`: ID de grupo de usuários do Amazon Cognito.
   - `AppClientId`: o ID do cliente da aplicação.

- `AppClientSecret`: o segredo do cliente da aplicação.
- `Region`: Sua AWS região.

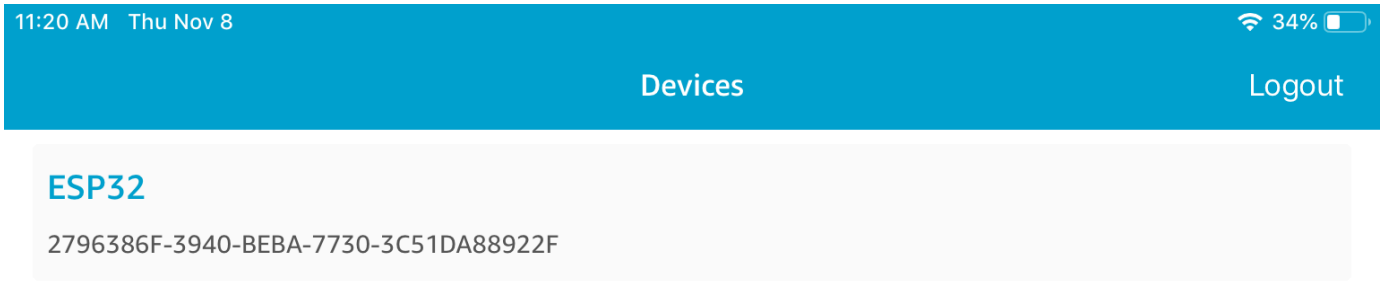
Para descobrir e estabelecer conexões seguras com seu microcontrolador por Bluetooth Low Energy

1. Para emparelhar seu microcontrolador e dispositivo móvel com segurança (etapa 6), você precisa de um emulador de terminal serial com recursos de entrada e saída (como). TeraTerm Configure o terminal para conectar-se à sua placa por uma conexão serial de acordo com as instruções em Instalação de um emulador de terminal.
2. Execute o projeto de demonstração de Bluetooth Low Energy no seu microcontrolador.
3. Execute a aplicação de demonstração do SDK móvel de Bluetooth Low Energy em seu dispositivo móvel.

   Para iniciar a aplicação de demonstração no Android para SDK a partir da linha de comando, execute o seguinte comando:

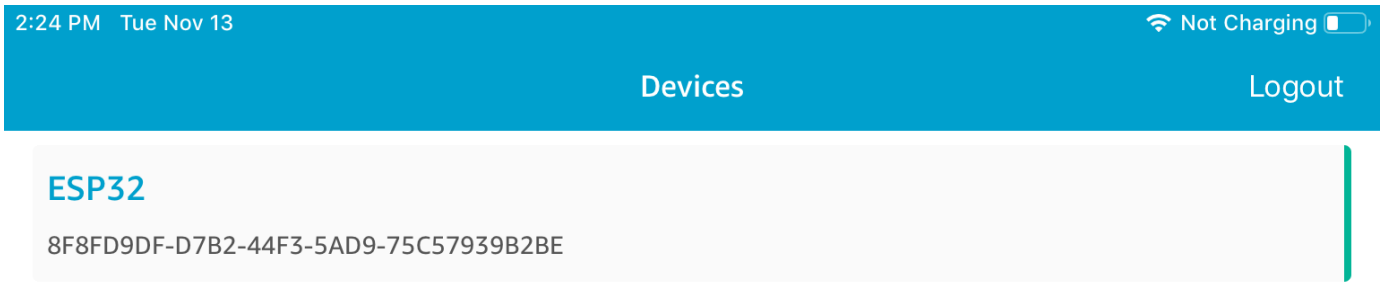
   ```
   $ ./gradlew installDebug
   ```

4. Confirme se o microcontrolador aparece em Devices (Dispositivos) na aplicação de demonstração do SDK móvel de Bluetooth Low Energy.

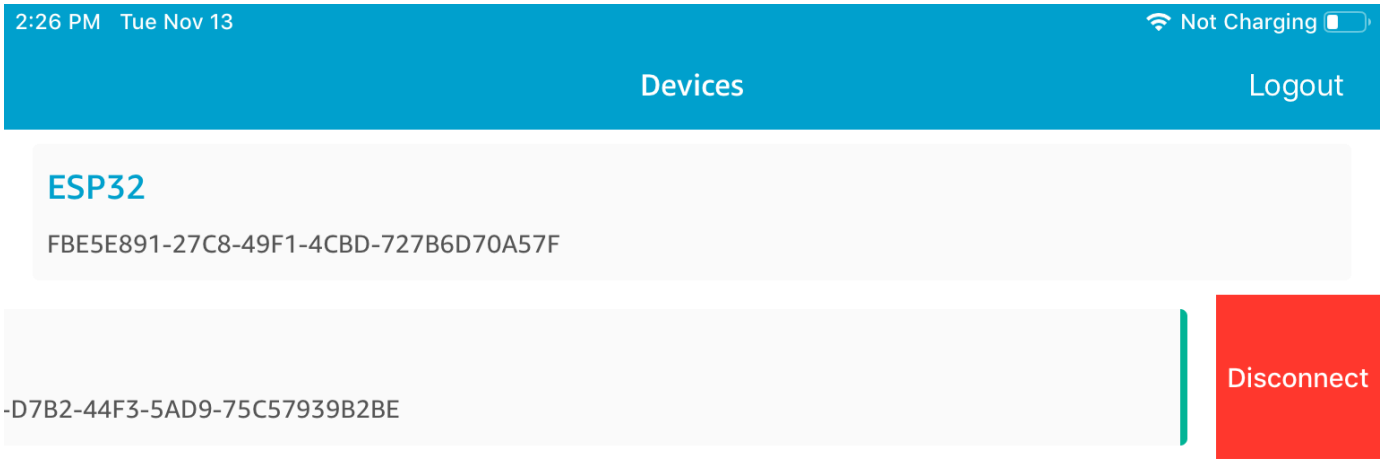


> **Note**
>
> Todos os dispositivos com o FreeRTOS e o serviço de informações de dispositivos (*freertos*`/.../device_information`) que estiverem no intervalo aparecerão na lista.

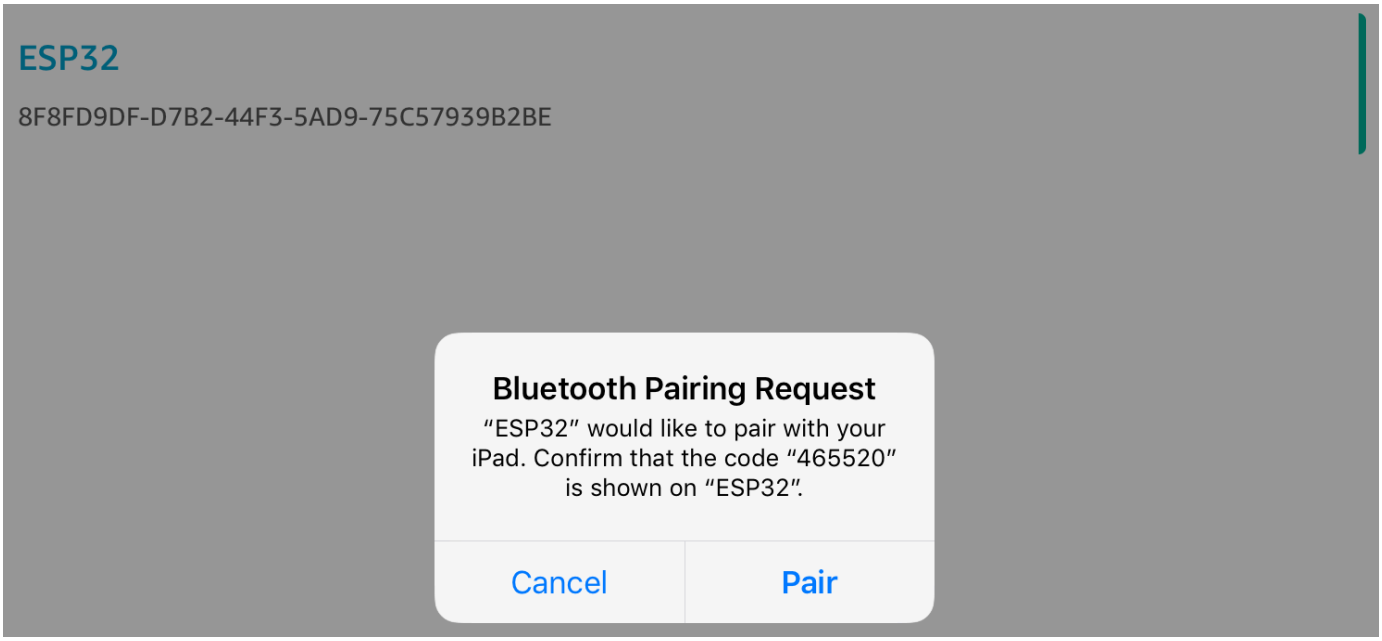
5. Escolha seu microcontrolador na lista de dispositivos. A aplicação estabelece uma conexão com a placa, e uma linha verde é exibida ao lado do dispositivo conectado.



Você pode se desconectar do microcontrolador arrastando a linha para a esquerda.



6. Se solicitado, emparelhe seu microcontrolador e dispositivo móvel.



Se o código para comparação numérica é o mesmo em ambos os dispositivos, pareie-os.

> **Note**
>
> A aplicação de demonstração do SDK móvel de Bluetooth Low Energy usa o Amazon Cognito para autenticação do usuário. Verifique se você configurou um usuário e banco de identidades do Amazon Cognito e se anexou as políticas do IAM às identidades autenticadas.

MQTT por Bluetooth Low Energy

Na demonstração do MQTT via Bluetooth Low Energy, seu microcontrolador publica mensagens AWS na nuvem por meio de um proxy MQTT.

Para inscrever-se em um tópico de demonstração do MQTT

1. Faça login no AWS IoT console.
2. No painel de navegação, escolha Teste e, em seguida, escolha cliente de teste MQTT para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***thing-name*****/example/topic1** e selecione Inscreva-se no tópico.

Se você usa Bluetooth Low Energy para emparelhar o microcontrolador com seu dispositivo móvel, as mensagens MQTT são roteadas por meio da aplicação de demonstração do SDK móvel de Bluetooth Low Energy em seu dispositivo móvel.

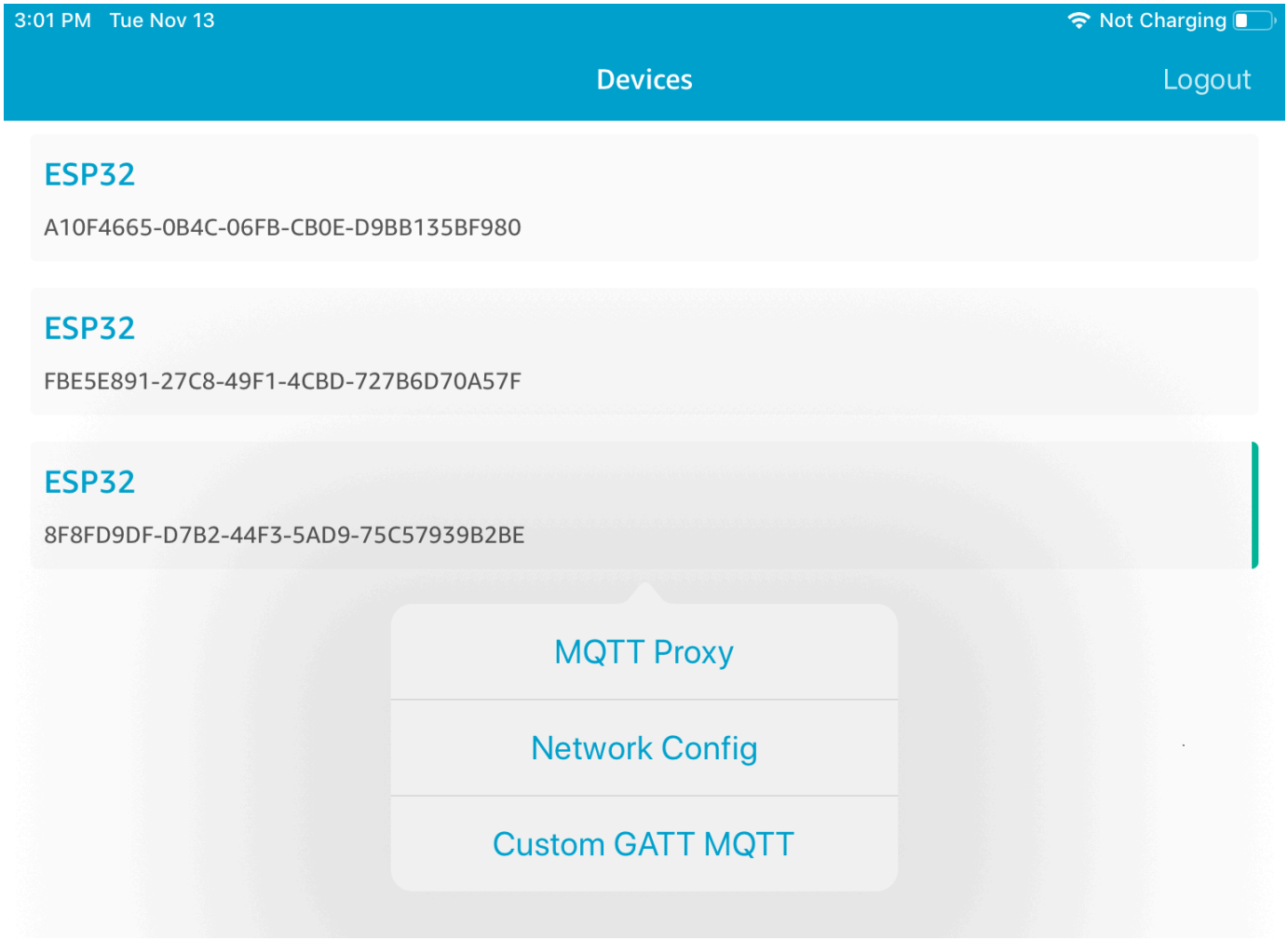
Como habilitar a demonstração por Bluetooth Low Energy

1. Abra `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` e defina `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`.
2. Abra `demos/include/aws_clientcredential.h` e configure `clientcredentialMQTT_BROKER_ENDPOINT` com o endpoint do AWS IoT broker. Configure `clientcredentialIOT_THING_NAME` com o nome da coisa para o dispositivo microcontrolador de BLE. O endpoint do AWS IoT broker pode ser obtido no AWS IoT console escolhendo Configurações no painel de navegação esquerdo ou por meio da CLI executando o comando:. `aws iot describe-endpoint --endpoint-type=iot:Data-ATS`

> **Note**
>
> O endpoint do AWS IoT broker e o nome da coisa devem estar na mesma região em que a identidade cognito e o grupo de usuários estão configurados.

Para executar a demonstração

1. Compile e execute o projeto de demonstração no seu microcontrolador.
2. Verifique se você pareou a placa e o dispositivo móvel usando o aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS.
3. Na lista Devices (Dispositivos) na aplicação móvel de demonstração, escolha seu microcontrolador e, depois, escolha MQTT Proxy (Proxy MQTT) para abrir as configurações do proxy MQTT.



4. Depois de habilitar o proxy MQTT, as mensagens MQTT são exibidas no tópico *thing-name*`/example/topic1` e os dados são impressos no terminal UART.

Provisionamento de Wi-Fi

O provisionamento de Wi-Fi é um serviço de Bluetooth Low Energy do FreeRTOS que permite enviar com segurança as credenciais de rede Wi-Fi de um dispositivo móvel para um microcontrolador por Bluetooth Low Energy. O código-fonte para o serviço de provisionamento de Wi-Fi pode ser encontrado em *freertos*`/.../wifi_provisioning`.

> **Note**
>
> Atualmente, a demonstração do Wi-Fi Provisioning é compatível com o Espressif ESP32- C. DevKit

Para habilitar a demonstração

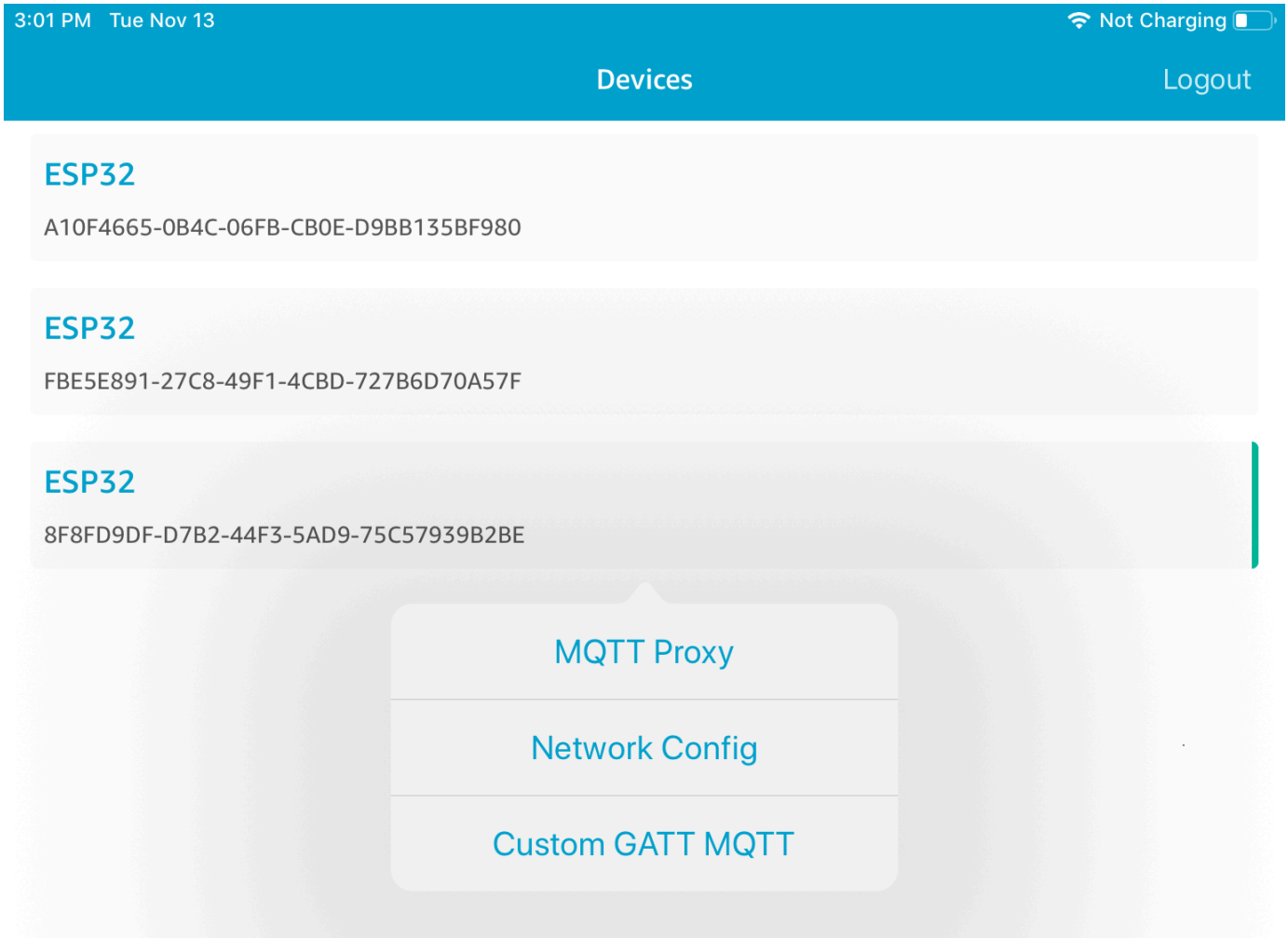
1. Habilite o serviço de provisionamento de Wi-Fi. Abra `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` e defina `#define IOT_BLE_ENABLE_WIFI_PROVISIONING` como `1` (em que *`vendor`* é o nome do fornecedor e *`board`* é o nome da placa que está sendo usada para executar as demonstrações).

   > **Note**
   >
   > O serviço de provisionamento de Wi-Fi é desabilitado por padrão.

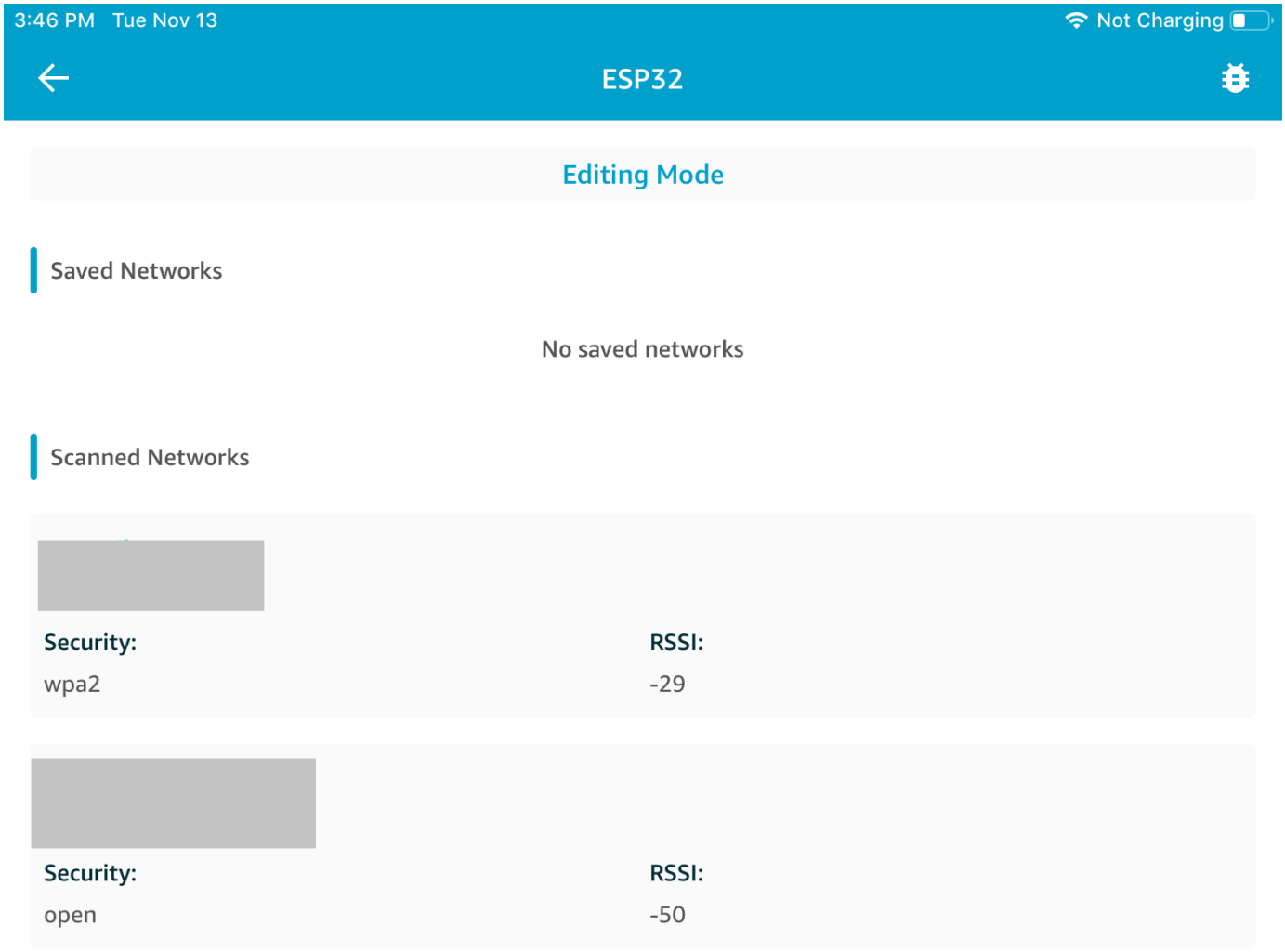
2. Configure o Gerenciador de rede para ativar tanto Bluetooth Low Energy quanto Wi-Fi.

Para executar a demonstração

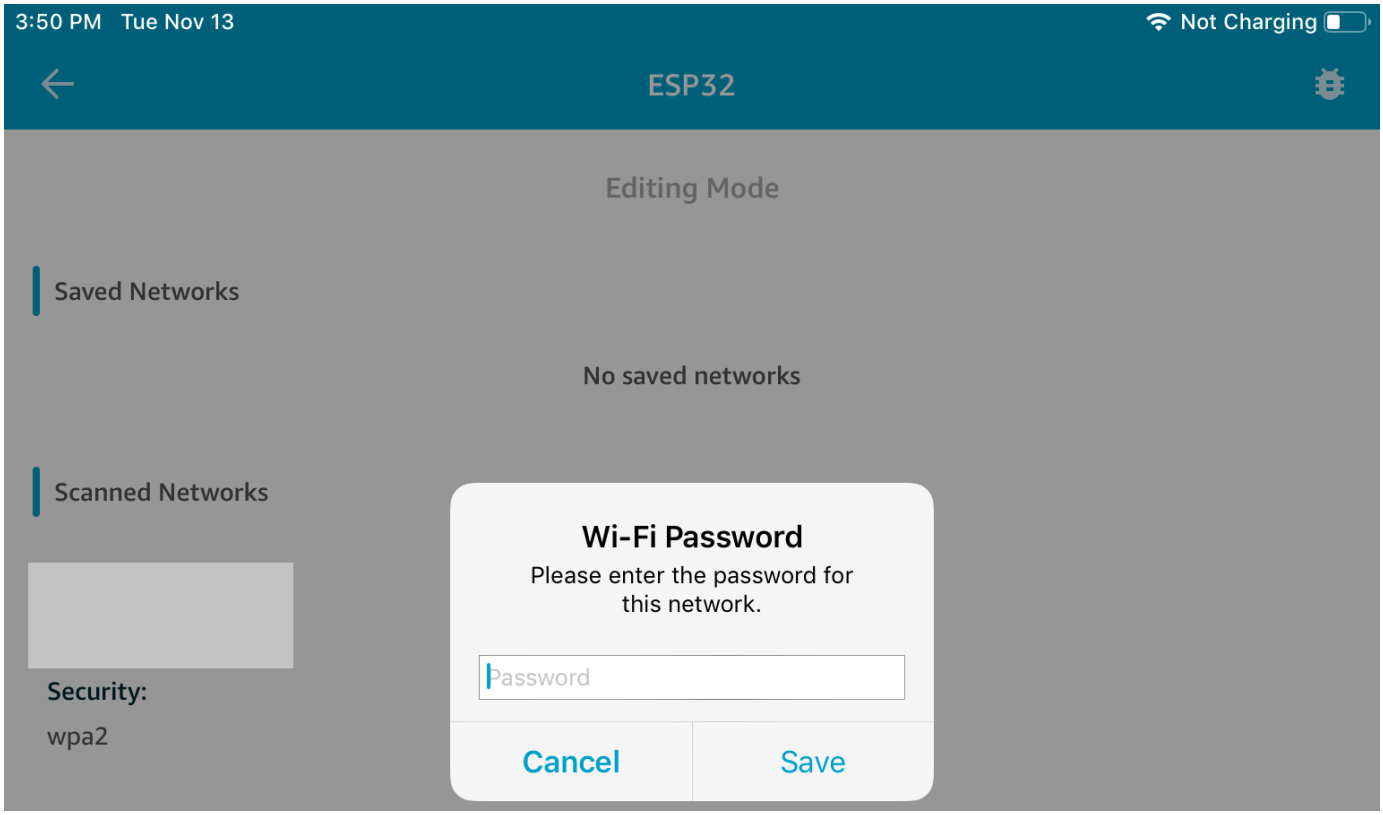
1. Compile e execute o projeto de demonstração no seu microcontrolador.
2. Verifique se você pareou o microcontrolador e o dispositivo móvel usando o aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS.
3. Na lista Devices (Dispositivos) na aplicação móvel de demonstração, escolha seu microcontrolador e, depois, escolha Network Config (Configuração de rede) para abrir as configurações de rede.



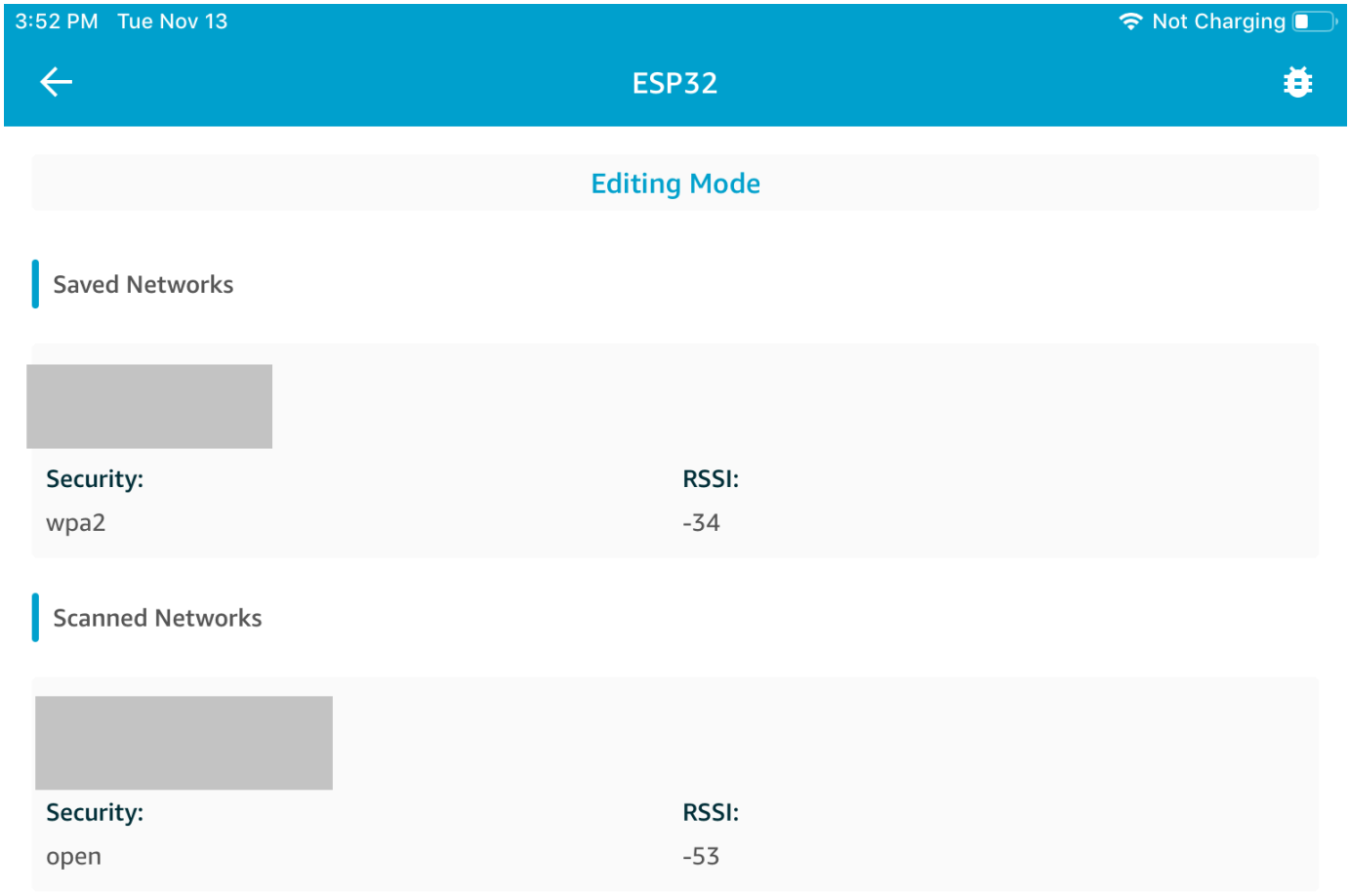
4. Depois que você escolher Network Config (Configuração de rede) para sua placa, o microcontrolador enviará uma lista de redes nas proximidades para o dispositivo móvel. As redes Wi-Fi disponíveis aparecem em uma lista em Scanned Networks (Redes verificadas).



Na lista Scanned Networks (Redes verificadas), escolha a rede e digite o SSID e a senha, se necessário.

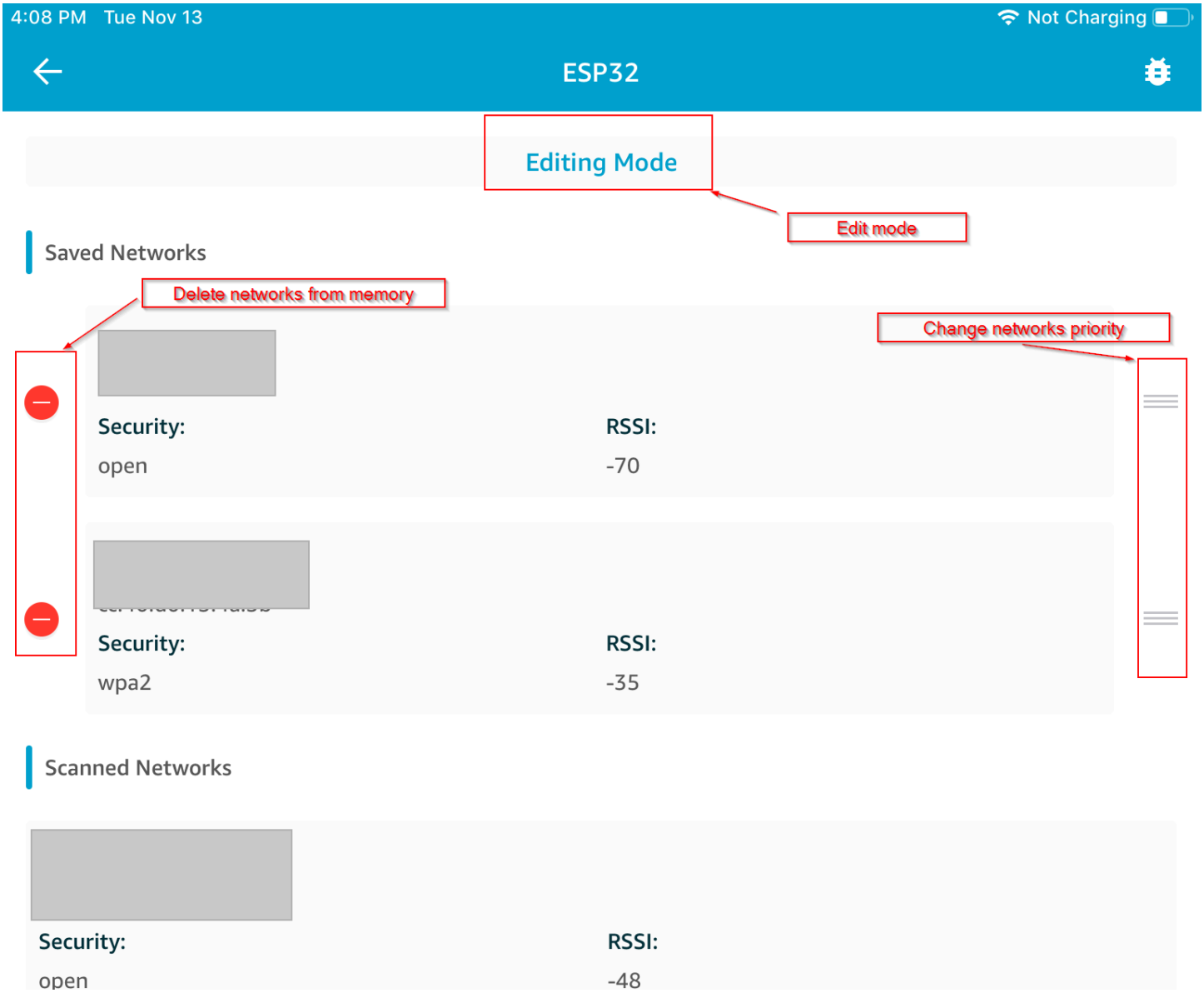


O microcontrolador conecta-se à rede e a salva. A rede é exibida nas Saved Networks (Redes salvas).



Você pode salvar várias redes na aplicação móvel de demonstração. Quando você reinicia a aplicação e a demonstração, o microcontrolador se conecta à primeira rede salva disponível, começando pela parte superior da lista de Saved Networks (Redes salvas).

Para alterar a ordem de prioridade das redes ou excluir redes, na página Network Configuration (Configuração de rede), escolha Editing Mode (Modo de edição). Para alterar a ordem de prioridade das redes, escolha o lado direito da rede que você deseja priorizar novamente e arraste-a para cima ou para baixo. Para excluir uma rede, escolha o botão vermelho no lado esquerdo da rede que você deseja excluir.



Servidor de atributos genéricos

Neste exemplo, uma aplicação do servidor de demonstração de recursos genéricos (GATT) no seu microcontrolador envia um valor de contador simples para o aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS.

Usando os SDKs móveis de Bluetooth Low Energy, você pode criar seu próprio cliente GATT para um dispositivo móvel que se conecta ao servidor GATT no seu microcontrolador e é executado em paralelo com a aplicação móvel de demonstração.

Para habilitar a demonstração

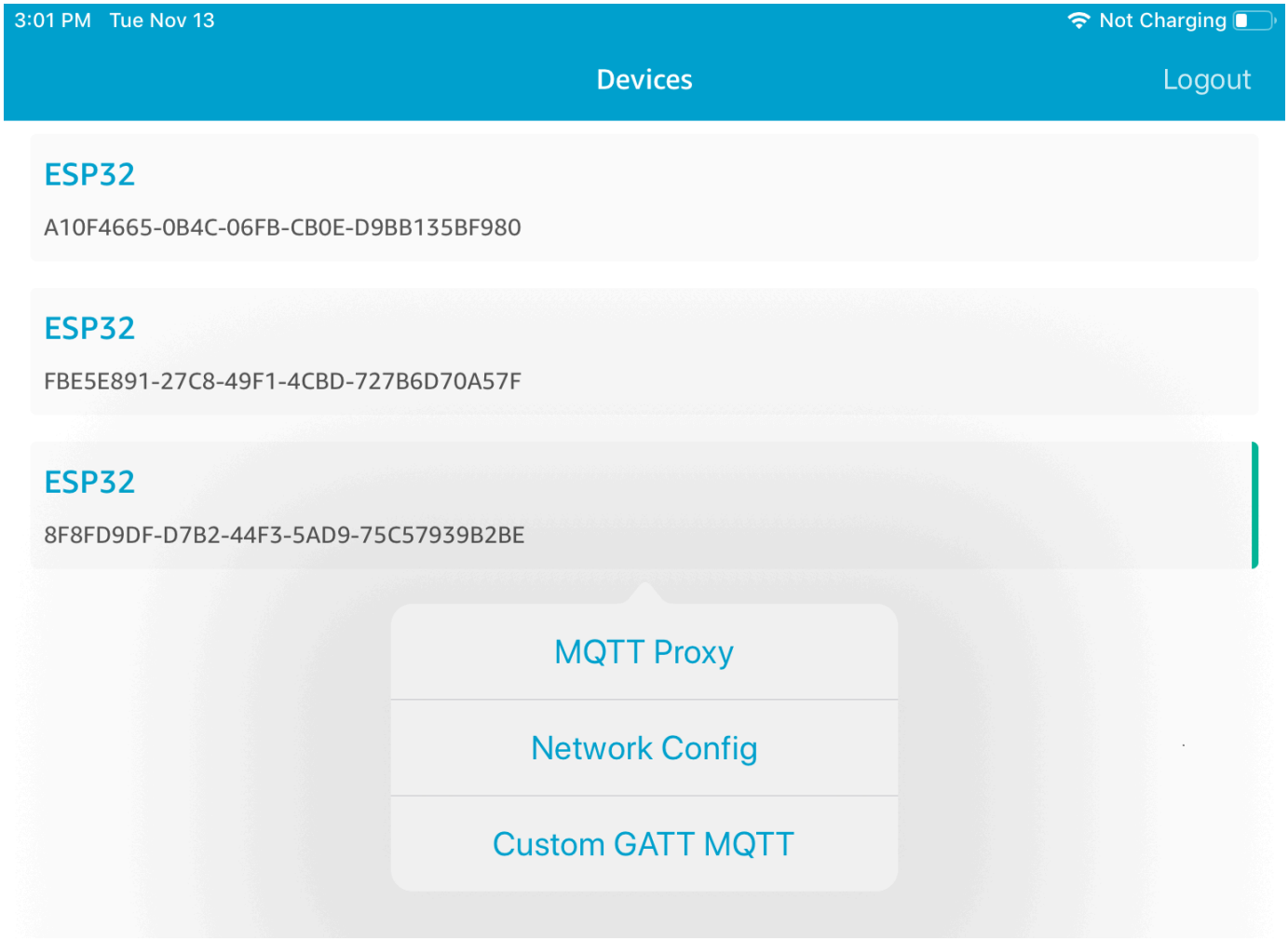
1. Ative a demonstração Bluetooth Low Energy GATT. Em `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` (em que *vendor* é o nome do fornecedor e *board* é o nome da placa que está sendo usada para executar as demonstrações), adicione `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` à lista de instruções de definição.

   > **Note**
   >
   > A demonstração Bluetooth Low Energy GATT é desabilitada por padrão.

2. Abra *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` e defina `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`.

Para executar a demonstração

1. Compile e execute o projeto de demonstração no seu microcontrolador.
2. Verifique se você pareou a placa e o dispositivo móvel usando o aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS.
3. Na lista Devices (Dispositivos) na aplicação, escolha sua placa e, depois, escolha MQTT Proxy (Proxy MQTT) para abrir as opções do proxy MQTT.



4. Retorne à lista Devices (Dispositivos), escolha sua placa e, depois, escolha Custom GATT MQTT (MQTT GATT personalizado) para abrir as opções de serviço de GATT personalizado.
5. Escolha Start Counter (Iniciar contador) para começar a publicar dados no tópico MQTT ***your-thing-name*/example/topic**.

   Depois de habilitar o proxy MQTT, o Hello World e as mensagens incrementais do contador aparecem no tópico *your-thing-name*/example/topic.

Bootloader de demonstração para o Microchip Curiosity PIC32MZEF

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto

FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Note

De acordo com a Microchip, estamos removendo o Curiosity PIC32MZEF (DM320104) da ramificação principal do repositório de Integração de Referência do FreeRTOS e não o carregaremos mais em novas versões. A Microchip emitiu um aviso oficial de que o PIC32MZEF (DM320104) não é mais recomendado para novos designs. Os projetos e o código-fonte do PIC32MZEF ainda podem ser acessados por meio das tags de lançamento anteriores. A Microchip recomenda que os clientes usem a placa de desenvolvimento Curiosity PIC32MZ-EF-2.0 (DM320209) para novos designs. A plataforma PIC32MZv1 ainda pode ser encontrada na versão v202012.00 do repositório de Integração de Referência do FreeRTOS. No entanto, não há mais suporte para a plataforma não na versão v202107.00 da Referência do FreeRTOS.

Este bootloader de demonstração implementa verificação de versão de firmware, verificação de assinatura criptográfica e testes automáticos da aplicação. Esses recursos oferecem suporte a atualizações de firmware over-the-air (OTA) para FreeRTOS.

A verificação de firmware inclui verificar a autenticidade e a integridade do novo firmware recebido pelo ar. O bootloader verifica a assinatura de criptografia da aplicação antes de inicializar. A demonstração usa o elliptic-curve digital signature algorithm (ECDSA – Algoritmo de assinatura digital de curva elíptica) sobre SHA-256. Os utilitários fornecidos podem ser usados para gerar uma aplicação assinada que pode ser atualizada no dispositivo.

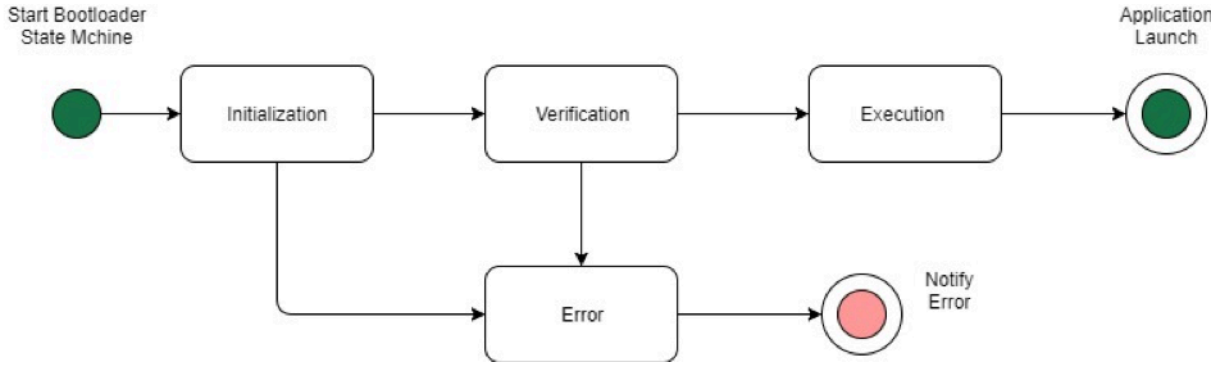
O bootloader é compatível com os seguintes recursos necessários para OTA:

- Mantém imagens de aplicação no dispositivo e alterna entre elas.
- Permite a execução do teste automático de uma imagem OTA recebida e faz reversão em caso de falha.
- Verifica a assinatura e versão da imagem de atualização OTA.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Estados de bootloader

O processo de bootloader é mostrado pela seguinte máquina de estado.



A tabela a seguir descreve os estados de bootloader.

Estado de bootloader	Descrição
Inicialização	O bootloader está no estado de inicialização.
Verificação	O bootloader está verificando as imagens presentes no dispositivo.
Executar Imagem	O bootloader está ativando a imagem seleciona da.
Executar Padrão	O bootloader está ativando a imagem padrão.
Erro	O bootloader está no estado de erro.

No diagrama anterior, tanto `Execute Image` quanto `Execute Default` são mostrados como o estado `Execution`.

Estado de execução do bootloader

O bootloader está no estado `Execution` e está pronto para iniciar a imagem verificada selecionada. Se a imagem a ser iniciada estiver no banco superior, os bancos serão trocados antes de executar a imagem porque a aplicação sempre é criada para o banco inferior.

Estado de execução padrão do bootloader

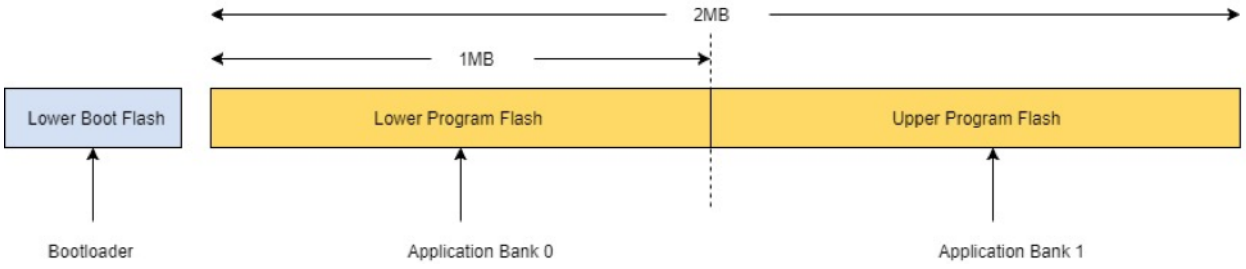
Se a opção de configuração para executar a imagem padrão estiver ativada, o bootloader iniciará a aplicação a partir de um endereço de execução padrão. Essa opção deve ser desativada, exceto durante a depuração.

Estado de erro de bootloader

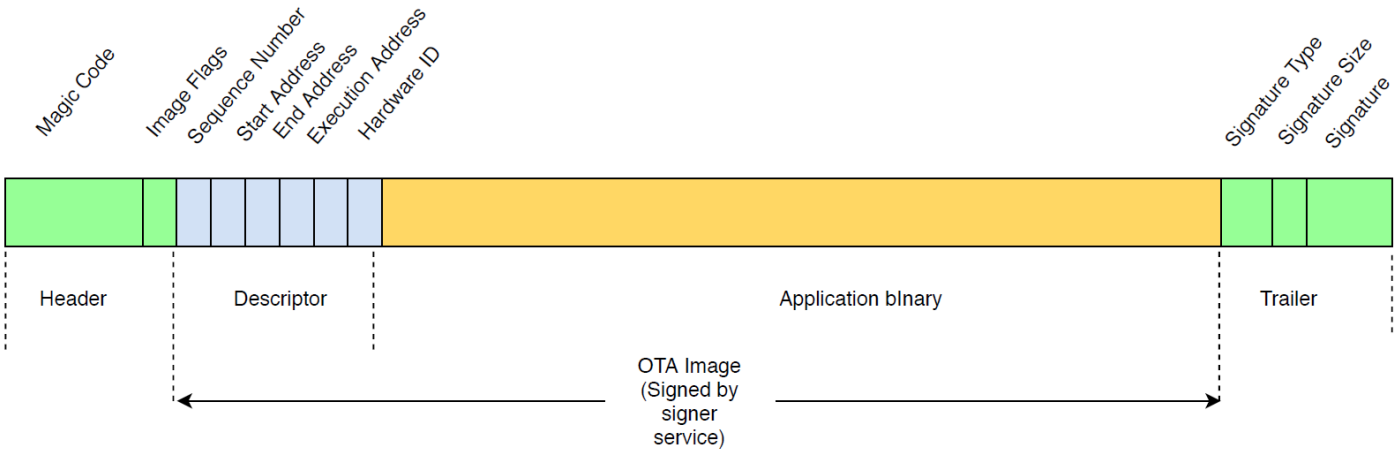
O bootloader está em um estado de erro e nenhuma imagem válida está presente no dispositivo. O bootloader deve notificar o usuário. A implementação padrão envia uma mensagem de log para o console e o LED pisca rapidamente na placa indefinidamente.

Dispositivo flash

A plataforma Microchip Curiosity PIC32MZEF contém um programa interno flash de dois megabytes (MB) dividido em dois bancos. Ele é compatível com troca de mapa de memória entre esses dois bancos e atualizações ao vivo. O bootloader de demonstração é programado em uma região flash separada de inicialização inferior.



Estrutura de imagem da aplicação



O diagrama mostra os componentes primários da imagem da aplicação armazenados em cada banco do dispositivo.

Componente	Tamanho (em bytes)
Cabeçalho da imagem	8 bytes
Descritor da imagem	24 bytes
Binário da aplicação	< 1 MB - (324)
Trailer	292 bytes

Cabeçalho da imagem

As imagens da aplicação no dispositivo devem começar com um cabeçalho que consiste em um código mágico e sinalizadores de imagem.

Campo de cabeçalho	Tamanho (em bytes)
Código mágico	7 bytes
Sinalizadores de imagem	1 byte

Código mágico

A imagem no dispositivo flash deve começar com um código mágico. O código mágico padrão é `@AFRTOS`. O bootloader verifica se um código mágico válido está presente antes de inicializar a imagem. Esta é a primeira etapa de verificação.

Sinalizadores de imagem

Os sinalizadores de imagem são usados para armazenar o status das imagens da aplicação. Os sinalizadores são usados no processo OTA. Os sinalizadores de imagem dos dois bancos determinam o estado do dispositivo. Se a imagem em execução estiver marcada como confirmação pendente, isso significa que o dispositivo está na fase de teste automático OTA. Mesmo que as imagens nos dispositivos sejam marcadas como válidas, elas passarão pelas mesmas etapas de verificação em cada inicialização. Se uma imagem for marcada como nova, o bootloader a marcará como confirmação pendente e a iniciará para teste automático após a verificação. O bootloader também inicializa e inicia o temporizador do watchdog para que, se a nova imagem OTA não passar no teste automático, o dispositivo seja reinicializado e o bootloader rejeite a imagem apagando-a e executa a imagem anterior válida.

O dispositivo pode ter apenas uma imagem válida. A outra imagem pode ser uma nova imagem OTA ou uma confirmação pendente (teste automático). Depois de uma atualização OTA bem-sucedida, a imagem antiga é apagada do dispositivo.

Status	Value	Descrição
Imagem nova	0xFF	A imagem da aplicação é nova e nunca foi executada.
Confirmação pendente	0xFE	A imagem da aplicação foi marcada para testar a execução.
Válido	0xFC	A imagem da aplicação foi marcada como válida e confirmada.
Inválido	0xF8	A imagem da aplicação foi marcada como inválida.

Descritor da imagem

A imagem da aplicação no dispositivo flash deve conter o seguinte descritor depois do cabeçalho da imagem. O descritor de imagem é gerado por um utilitário pós-compilação que usa os arquivos de configuração (`ota-descriptor.config`) para gerar o descritor apropriado e o anexa ao binário da aplicação. A saída dessa etapa pós-compilação é a imagem binária que pode ser usada para OTA.

Campo do descritor	Tamanho (em bytes)
Número de sequência	4 bytes
Endereço inicial	4 bytes
Endereço final	4 bytes
Endereço de execução	4 bytes
ID de hardware	4 bytes
Reservado	4 bytes

Número de sequência

O número sequencial deve ser aumentado antes de criar uma nova imagem OTA. Consulte o arquivo `ota-descriptor.config`. O bootloader usa esse número para determinar a imagem para inicializar. Os valores válidos variam de 1 a 4294967295.

Endereço inicial

O endereço inicial da imagem da aplicação no dispositivo. Como o descritor de imagem é acrescentado ao binário da aplicação, esse endereço é o início do descritor de imagem.

Endereço final

O endereço final da imagem da aplicação no dispositivo, excluindo o trailer da imagem.

Endereço de execução

O endereço de execução da imagem.

ID de hardware

Um ID de hardware exclusivo usado pelo bootloader para verificar se a imagem OTA foi criada para a plataforma correta.

Reservado

Isso está reservado para uso futuro.

Trailer de imagem

O trailer da imagem é anexado ao binário da aplicação. Ele contém a string de tipo de assinatura, tamanho da assinatura e assinatura da imagem.

Campo Trailer	Tamanho (em bytes)
Tipo de assinatura	32 bytes
Tamanho da assinatura	4 bytes
Assinatura	256 bytes

Tipo de assinatura

O tipo de assinatura é uma string que representa o algoritmo de criptografia que está sendo usado e serve como um marcador para o trailer. O bootloader é compatível com o algoritmo de assinatura digital de curva elíptica (ECDSA). O padrão é sig-sha256-ecdsa.

Tamanho da assinatura

O tamanho da assinatura de criptografia, em bytes.

Assinatura

A assinatura de criptografia do binário da aplicação prefixado com o descritor de imagem.

Configuração de bootloader

As opções de configuração básica do bootloader são fornecidas em *freertos*`/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h`. Algumas opções são fornecidas apenas para fins de depuração.

Ativar início padrão

Habilita a execução da aplicação a partir do endereço padrão e deve ser habilitado para depuração apenas. A imagem é executada a partir do endereço padrão sem nenhuma verificação.

Habilitar verificação de assinatura de criptografia

Habilita a verificação de assinatura criptográfica na inicialização. As imagens com falha são apagadas do dispositivo. Essa opção é fornecida apenas para fins de depuração e deve permanecer habilitada na produção.

Apagar imagem inválida

Permite apagar um banco inteiro caso a verificação da imagem falhe nesse banco. A opção é fornecida para depuração e deve permanecer habilitada na produção.

Ativar verificação de ID de hardware

Ativa a verificação do ID de hardware no descritor da imagem OTA e do ID de hardware programado no bootloader. Isso é opcional e pode ser desativado se a verificação do ID de hardware não for necessária.

Ativar verificação de endereço

Habilita a verificação dos endereços de execução, inicial e final no descritor da imagem OTA. Recomendamos manter essa opção ativada.

Criação do bootloader

O bootloader de demonstração é incluído como um projeto carregável no projeto `aws_demos` localizado em *`freertos`*`/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/` no repositório de código-fonte do FreeRTOS. Quando o projeto `aws_demos` é criado, ele cria o bootloader primeiro, seguido pela aplicação. A saída final é uma imagem hexadecimal unificada incluindo o bootloader e a aplicação. O utilitário `factory_image_generator.py` é fornecido para gerar uma imagem hexadecimal unificada com assinatura de criptografia. Os scripts de utilitário do bootloader estão localizados em *`freertos`*`/demos/ota/bootloader/utility/`.

Etapa de pré-compilação de bootloader

Esta etapa de pré-compilação executa um script chamado `codesigner_cert_utility.py` que extrai a chave pública do certificado de assinatura de código e gera um arquivo de cabeçalho C

que contém a chave pública no formato codificado Abstract Syntax Notation One (ASN.1). Esse cabeçalho é compilado no projeto do bootloader. O cabeçalho gerado contém duas constantes: uma matriz da chave pública e o comprimento da chave. O projeto do bootloader também pode ser criado sem `aws_demos` e pode ser depurado como aplicação normal.

AWS IoT Device Defender demonstração

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

Esta demonstração mostra como usar a biblioteca do AWS IoT Device Defender para se conectar AWS IoT Device Defendera. A demonstração usa a biblioteca CoreMQTT para estabelecer uma conexão MQTT via TLS (autenticação mútua) com o AWS IoT MQTT Broker e a biblioteca CoreJSON para validar e analisar as respostas recebidas do serviço. AWS IoT Device Defender A demonstração mostra como criar um relatório em formato JSON usando métricas coletadas do dispositivo e como enviar o relatório construído para o AWS IoT Device Defender serviço. A demonstração também mostra como registrar uma função de retorno de chamada na biblioteca CoreMQTT para lidar com as respostas do AWS IoT Device Defender serviço e confirmar se um relatório enviado foi aceito ou rejeitado.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Funcionalidade

Esta demonstração cria uma única tarefa de aplicativo que demonstra como coletar métricas, criar um relatório do Device Defender no formato JSON e enviá-lo ao AWS IoT Device Defender serviço por meio de uma conexão MQTT segura com o MQTT Broker. AWS IoT A demonstração inclui

as métricas de rede padrão, bem como métricas personalizadas. Para métricas personalizadas, a demonstração inclui:

- Uma métrica chamada "`task_numbers`" que é uma lista de IDs de tarefas do FreeRTOS. O tipo dessa métrica é "lista de números".
- Uma métrica chamada "`stack_high_water_mark`" que é a marca d'água no topo da pilha para a tarefa da aplicação de demonstração. O tipo dessa métrica é "números".

A maneira de coletar métricas de rede depende da pilha TCP/IP em uso. Para FreeRTOS+TCP e configurações LWIP compatíveis, fornecemos implementações de coleta de métricas que coletam métricas reais do dispositivo e as enviam no relatório. AWS IoT Device Defender Você pode encontrar as implementações do FreeRTOS+TCP e do LWIP em. GitHub

Para placas que usam uma outra pilha TCP/IP, fornecemos as definições de esboço das funções de coleta de métricas que retornam zeros para todas as métricas de rede. Implemente as funções em *`freertos`*`/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c` na sua pilha de rede para a pilha de rede enviar métricas reais. O arquivo também está disponível no GitHubsite.

Para o ESP32, a configuração padrão do lwIP não usa bloqueio principal, sendo assim, a demonstração usa métricas esboçadas. Se desejar usar a implementação da coleção de métricas lwIP de referência, defina as seguintes macros em `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
```

Veja um exemplo de saída quando você executa a demonstração a seguir.

```
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-------
```

Se sua placa não estiver usando FreeRTOS+TCP ou uma configuração lwIP compatível, a saída será semelhante à seguinte.

```
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-------
```

O código-fonte da demonstração está em seu download no *freertos*/demos/device_defender_for_aws/ diretório ou no GitHubsite.

Inscrevendo-se em tópicos AWS IoT Device Defender

A função subscribeToDefenderTópicos se inscreve nos tópicos do MQTT sobre os quais as respostas aos relatórios publicados do Device Defender serão recebidas. Ela usa a macro `DEFENDER_API_JSON_ACCEPTED` para criar a sequência de tópicos na qual as respostas aos relatórios aceitos do Device Defender são recebidas. Ela usa a macro `DEFENDER_API_JSON_REJECTED` para criar a sequência de tópicos na qual as respostas aos relatórios rejeitados do Device Defender serão recebidas.

Coleta de métricas de dispositivos

A collectDeviceMetricsfunção reúne métricas de rede usando as funções definidas em`metrics_collector.h`. As métricas coletadas são o número de bytes e pacotes enviados e recebidos, as portas TCP abertas, as portas UDP abertas e as conexões TCP estabelecidas.

Gerando o AWS IoT Device Defender relatório

A função generateDeviceMetricsReport gera um relatório do Device Defender usando a função definida em`report_builder.h`. Essa função pega as métricas de rede e um buffer, cria um documento JSON no formato esperado AWS IoT Device Defender e o grava no buffer fornecido. O formato do documento JSON esperado por AWS IoT Device Defender é especificado nas métricas do lado do dispositivo no Guia do desenvolvedor.AWS IoT

Publicando o AWS IoT Device Defender relatório

O AWS IoT Device Defender relatório é publicado no tópico MQTT para publicação de relatórios JSON AWS IoT Device Defender . O relatório é construído usando a macro`DEFENDER_API_JSON_PUBLISH`, conforme mostrado neste trecho de código no GitHub site.

Retorno de chamada tratar respostas

A função publishCallback trata as mensagens MQTT de publicação recebidas. Ele usa a `Defender_MatchTopic` API da AWS IoT Device Defender biblioteca para verificar se a mensagem MQTT recebida é do AWS IoT Device Defender serviço. Se a mensagem for do AWS IoT Device Defender serviço, ela analisará a resposta JSON recebida e extrairá o ID do relatório na resposta. Em seguida, será verificado se o ID do relatório é o mesmo que foi enviado no relatório.

Aplicativo de demonstração do AWS IoT Greengrass Discovery V1

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Antes de executar a demonstração do AWS IoT Greengrass Discovery para FreeRTOS, é necessário configurar a AWS, o AWS IoT Greengrass, e o AWS IoT. Paga configurar a AWS, siga as instruções em Configurando sua AWS conta e permissões. Para configurar o AWS IoT Greengrass, é necessário criar um grupo do Greengrass e adicionar um núcleo do Greengrass. Para obter mais informações sobre a configuração do AWS IoT Greengrass, consulte Conceitos básicos do AWS IoT Greengrass.

Depois de configurar a AWS e o AWS IoT Greengrass, será necessário configurar algumas permissões adicionais para o AWS IoT Greengrass.

Para configurar permissões do AWS IoT Greengrass

1. Navegue até o console do IAM.
2. No painel de navegação, selecione Roles (Funções_ e, então, procure e selecione Greengrass_ServiceRole.
3. Selecione Attach policies (Anexar políticas), selecione AmazonS3FullAccess e AWSIoTFullAccess e selecione Attach policy (Anexar política).
4. Navegue até o console do AWS IoT.
5. No painel de navegação, selecione Greengrass, selecione Groups (Grupos) e selecione o grupo do Greengrass criado anteriormente.
6. Selecione Settings (Configurações) e Add role (Adicionar função).
7. Selecione Greengrass_ServiceRole e Save (Salvar).

Conecte a placa do AWS IoT e configure a demonstração do FreeRTOS.

1. Registrando sua placa MCU com AWS IoT

Depois de registrar a placa, será necessário criar e anexar uma nova política do Greengrass ao certificado do dispositivo.

Para criar uma política do AWS IoT Greengrass

1. Navegue até o console do AWS IoT.
2. No painel de navegação, escolha Secure (Seguro), Policies (Políticas) e Create (Criar).
3. Insira um nome para identificar a política.
4. Na seção Add statements (Adicionar instruções), escolha Advanced mode (Modo avançado). Copie e cole o seguinte JSON na janela do editor de política:

   ```
   {
         "Effect": "Allow",
         "Action": [
           "greengrass:*"
         ],
         "Resource": "*"
   }
   ```

   Essa política concede permissões ao AWS IoT Greengrass para todos os recursos.
5. Escolha Create (Criar).

Para anexar a política do AWS IoT Greengrass ao certificado do dispositivo

1. Navegue até o console do AWS IoT.
2. No painel de navegação, selecione Manage (Gerenciar), selecione Things (Coisas) e selecione a coisa criada anteriormente.
3. Selecione Security (Segurança) e selecione o certificado anexado ao dispositivo.
4. Selecione Policies (Políticas), selecione Actions (Ações) e selecione Attach Policy (Anexar política).
5. Encontre e selecione a política do Greengrass criada anteriormente e selecione Attach (Anexar).

2. Fazer download do FreeRTOS

> **Note**
>
> Se você estiver fazendo download do FreeRTOS, a partir do console dele, escolha Conectar-se ao AWS IoT Greengrass – ***Plataforma*** em vez de Conectar-se ao AWS IoT – ***Plataforma***.

3. Configuração das demonstrações do FreeRTOS.

   Abra *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, comente #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED e defina CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED.

Depois de configurar a AWS IoT e o AWS IoT Greengrass, fazer o download e configurar o FreeRTOS, você poderá compilar, instalar e executar a demonstração do Greengrass no seu dispositivo. Para configurar o ambiente de desenvolvimento de hardware e software da placa, siga as instruções no Guias de conceitos básicos específicos da placa.

A demonstração do Greengrass publica uma série de mensagens no núcleo do Greengrass e no cliente MQTT da AWS IoT. Para visualizar as mensagens no cliente MQTT do AWS IoT, abra o console AWS IoT, selecione Testar, escolha cliente de teste MQTT e adicione uma assinatura a `freertos/demos/ggd`.

No cliente MQTT, você deve ver estas strings:

```
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 uma instância do Amazon EC2

Se você estiver trabalhando com uma instância do Amazon EC2

1. Localize o DNS público (IPv4) associado à sua instância do Amazon EC2 – acesse o console do Amazon EC2 e, no painel de navegação esquerdo, escolha Instâncias. Escolha sua instância do Amazon EC2 e escolha o painel Descrição. Procure a entrada para o DNS público (IPv4) e anote-o.

2. Localize a entrada para Grupos de segurança e escolha o grupo de segurança anexado à sua instância do Amazon EC2.
3. Escolha a guia Regras de entrada e escolha Editar regras de entrada e adicione as seguintes regras.

   Regras de entrada

Type	Protocolo	Intervalo de portas	Origem	Descrição (opcional)
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
TCP personalizado	TCP	8883	0.0.0.0/0	Comunicações MQTT
TCP personalizado	TCP	8883	::/0	Comunicações MQTT
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::0/0	-
Todos os ICMP - IPv4	ICMP	Tudo	0.0.0.0/0	-
Todos os ICMP - IPv4	ICMP	Tudo	::0/0	-

4. No console do AWS IoT, escolha Greengrass, Grupos e escolha o grupo do Greengrass criado anteriormente. Escolha Settings (Configurações). Altere a Detecção de conexão local para Gerenciar manualmente as informações de conexão.
5. No painel de navegação, escolha Núcleos e selecione o núcleo do grupo.
6. Escolha Conectividade e verifique se você tem somente um endpoint principal (exclua todos os outros) e se ele não é um endereço IP (porque está sujeito a alterações). A melhor opção é usar o DNS público (IPv4) anotado na primeira etapa.

7. Adicione a coisa da IoT do FreeRTOS criada para o grupo do GG.

   a. Escolha a seta para trás para retornar à página do grupo do AWS IoT Greengrass. No painel de navegação, escolha Dispositivos e Adicionar dispositivo.

   b. Escolha Selecionar uma coisa da IoT. Escolha o dispositivo e escolha Concluir.

8. Adicione as assinaturas necessárias na página Grupo do Greengrass, escolha Assinaturas, escolha Adicionar assinatura e insira as informações conforme mostrado aqui.

   Assinaturas

Origem	Destino	Tópico
TIGG1	IoT Cloud	freertos/demos/ggd

   Onde "Fonte" é o nome dado à coisa do AWS IoT criada no console do AWS IoT quando você registrou sua placa - "TIGG1" no exemplo dado aqui.

9. Inicie uma implantação do seu grupo do AWS IoT Greengrass e verifique se a implantação foi bem-sucedida. Agora, você deve conseguir executar com êxito a demonstração do AWS IoT Greengrass Discovery.

AWS IoT Greengrass V2

Compatibilidade com dispositivos do AWS IoT Greengrass V2

O suporte do AWS IoT Greengrass V2 para dispositivos cliente é compatível com versões anteriores de AWS IoT Greengrass V1. Você pode conectar dispositivos cliente FreeRTOS a dispositivos V2 core sem alterar o código do aplicativo. Para permitir que dispositivos cliente se conectem a um dispositivo de núcleo V2, faça o seguinte.

- Implante o software Greengrass no dispositivo de núcleo do Greengrass. Consulte Conectar dispositivos cliente a dispositivos de núcleo para conectar um dispositivo ao AWS IoT Greengrass V2.
- Para retransmitir mensagens (incluindo funções Lambda) entre dispositivos cliente, serviço de nuvem do AWS IoT Core e componentes do Greengrass, implante e configure o componente de ponte MQTT.
- Implante o componente detector de IP para detectar automaticamente as informações de conectividade ou gerenciar manualmente os endpoints.

- Consulte Interagir com dispositivos AWS IoT locais para obter mais informações.

Para obter mais detalhes, consulte a documentação da AWS sobre a execução de aplicativos do AWS IoT Greengrass V1 no AWS IoT Greengrass V2.

Demonstrações de coreHTTP

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Essas demonstrações podem ajudar você a aprender como usar a biblioteca coreHTTP.

Tópicos

- Demonstração de autenticação mútua da coreHTTP
- Demonstração de upload da coreHTTP básica do Amazon S3
- Demonstração de download de coreHTTP básica do S3
- Demonstração de threads múltiplas básica da coreHTTP

Demonstração de autenticação mútua da coreHTTP

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

O projeto de demonstração da coreHTTP (autenticação mútua) mostra como estabelecer uma conexão com um servidor HTTP usando TLS com autenticação mútua entre o cliente e o servidor.

Esta demonstração usa uma implementação de interface de transporte baseada em mbedTLS para estabelecer uma conexão TLS autenticada pelo servidor e pelo cliente e demonstra um fluxo de trabalho de resposta à solicitação em HTTP.

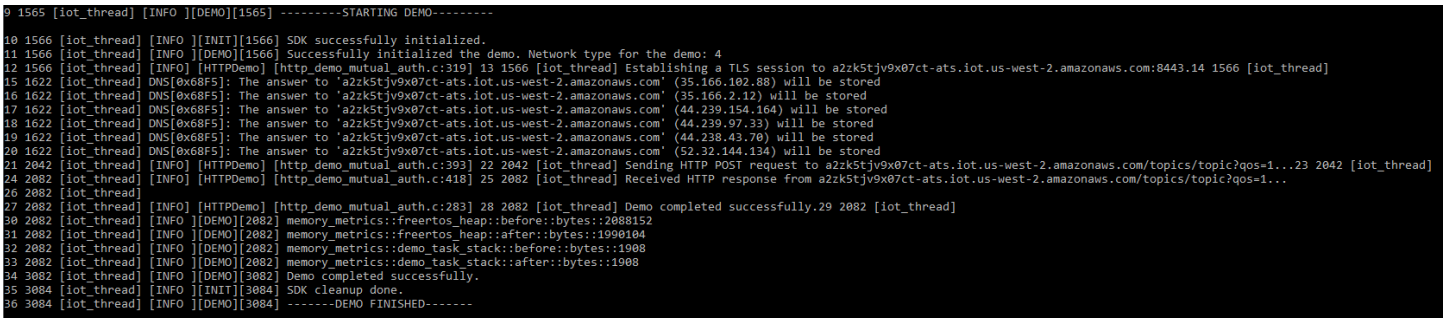
> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Funcionalidade

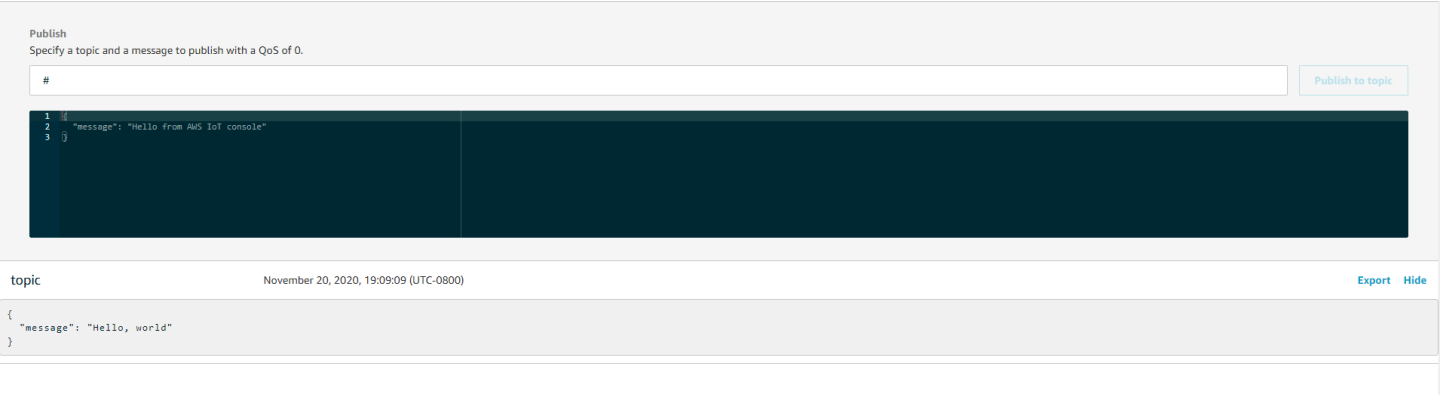
Essa demonstração cria uma tarefa de aplicação única com exemplos que mostram como concluir o seguinte:

- Conecte-se ao servidor HTTP no AWS IoT endpoint.
- Enviar uma solicitação POST.
- Receber a resposta.
- Desconectar do servidor.

Depois de concluir essas etapas, a demonstração gera uma saída semelhante à da captura de tela a seguir.



O AWS IoT console gera uma saída semelhante à captura de tela a seguir.



Organização de código-fonte

O arquivo fonte de demonstração tem um nome `http_demo_mutual_auth.c` e pode ser encontrado no *`freertos`*`/demos/coreHTTP/` diretório e no GitHubsite.

Conectando-se ao servidor AWS IoT HTTP

A função connectToServerWithBackoffRetries tenta fazer uma conexão TLS autenticada mutuamente com o servidor HTTP. AWS IoT Se a conexão falhar, ela tentará novamente após um tempo limite. O valor do tempo limite aumenta exponencialmente até que o número máximo de tentativas ou o valor do tempo limite seja atingido. A função `RetryUtils_BackoffAndSleep` fornece valores de tempo limite aumentando exponencialmente e retorna `RetryUtilsRetriesExhausted` quando o número máximo de tentativas foi atingido. A função `connectToServerWithBackoffRetries` retorna um status de falha se a conexão TLS com o operador não puder ser estabelecida após o número configurado de tentativas.

Envio de uma solicitação HTTP e recebimento da resposta

A função prvSendHttpRequest demonstra como enviar uma solicitação POST para o servidor AWS IoT HTTP. Para obter mais informações sobre como fazer uma solicitação para a API REST em AWS IoT, consulte Protocolos de comunicação do dispositivo - HTTPS. A resposta é recebida com a mesma chamada de API da coreHTTP, `HTTPClient_Send`.

Demonstração de upload da coreHTTP básica do Amazon S3

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto

FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

Este exemplo demonstra como enviar uma solicitação PUT ao servidor HTTP do Amazon Simple Storage Service (Amazon S3) e fazer upload de um arquivo pequeno. Ele também executa uma solicitação GET para verificar o tamanho do arquivo após o upload. Este exemplo usa uma interface de transporte de rede que usa mbedTLS para estabelecer uma conexão mutuamente autenticada entre um cliente de dispositivo de IoT executando coreHTTP e o servidor HTTP do Amazon S3.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Thread único versus thread múltiplo

Existem dois modelos de uso da coreHTTP, com thread único e com threads múltiplos (multitarefas). Embora a demonstração nesta seção execute a biblioteca HTTP em um tópico, ela realmente demonstra como usar a coreHTTP em um ambiente de tópico único. A API HTTP é usada somente por uma tarefa nesta demonstração. Embora as aplicações com thread único devam chamar repetidamente a biblioteca HTTP, as aplicações com threads múltiplos podem enviar solicitações HTTP em segundo plano em uma tarefa de agente (ou daemon).

Organização de código-fonte

O arquivo fonte de demonstração tem um nome `http_demo_s3_upload.c` e pode ser encontrado no *`freertos`*`/demos/coreHTTP/` diretório e no GitHubsite.

Configuração de conexão do servidor HTTP do Amazon S3

Esta demonstração usa uma URL pré-assinada para se conectar ao servidor HTTP do Amazon S3 e autorizar o acesso ao objeto para download. A conexão TLS do servidor HTTP do Amazon S3 usa somente a autenticação do servidor. No nível da aplicação, o acesso ao objeto é autenticado com os parâmetros na consulta de URL pré-assinada. Siga as etapas abaixo para configurar sua conexão com a AWS.

1. Configure uma AWS conta:

   a. Se ainda não o fez, crie uma AWS conta.

   b. As contas e permissões são definidas usando AWS Identity and Access Management (IAM). O IAM é usado para gerenciar as permissões de cada usuário em sua conta. Por padrão, um usuário não tem permissões até que sejam concedidas pelo proprietário raiz.

      i. Para adicionar um usuário à sua AWS conta, consulte o Guia do usuário do IAM.

      ii. Conceda permissão à sua AWS conta para acessar os FreeRTOS adicionando esta AWS IoT política:

      - Amazon S3 FullAccess

2. Criar um bucket no Amazon S3 seguindo as etapas em Como criar um bucket do S3? no Guia do usuário do Amazon Simple Storage Service.
3. Faça upload de um arquivo no Amazon S3 seguindo as etapas em Como fazer upload de arquivos e pastas em um bucket do S3?.
4. Gere um URL pré-assinado usando o script localizado no arquivo `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`.

   Para obter instruções de uso, consulte o arquivo `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`.

Funcionalidade

A demonstração primeiro se conecta ao servidor HTTP do Amazon S3 com a autenticação do servidor TLS. Em seguida, cria uma solicitação HTTP para carregar os dados especificados em `democonfigDEMO_HTTP_UPLOAD_DATA`. E verifica se o arquivo foi carregado com êxito solicitando o tamanho do arquivo, depois de fazer upload do arquivo. O código-fonte da demonstração pode ser encontrado no GitHubsite.

Conexão ao servidor HTTP do Amazon S3

A função connectToServerWithBackoffRetries tenta fazer uma conexão TCP com o servidor HTTP. Se a conexão falhar, ela tentará novamente após um tempo limite. O valor do tempo limite aumentará exponencialmente até que o número máximo de tentativas ou o valor do tempo limite seja atingido. A função `connectToServerWithBackoffRetries` retorna um status de falha se a conexão TCP com o servidor não puder ser estabelecida após o número configurado de tentativas.

A função `prvConnectToServer` demonstra como estabelecer uma conexão com o servidor HTTP do Amazon S3 usando somente a autenticação do servidor. Ela usa a interface de transporte baseada em mbedTLS que é implementada no arquivo `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c`. A definição de `prvConnectToServer` pode ser encontrada no GitHubsite.

Upload de dados

A função `prvUploadS3ObjectFile` demonstra como criar uma solicitação PUT e especificar o arquivo para fazer upload. O bucket do Amazon S3 para o qual é feito o upload do arquivo e o nome desse arquivo são especificados na URL pré-assinada. Para economizar memória, o mesmo buffer é usado para os cabeçalhos de solicitação e para receber a resposta. A resposta é recebida de forma síncrona usando a função de API `HTTPClient_Send`. Um código de status de resposta `200 OK` é esperado do servidor HTTP do Amazon S3. Qualquer outro código de status é um erro.

O código-fonte do `prvUploadS3ObjectFile()` pode ser encontrado no GitHubsite.

Verificação do upload

A função `prvVerifyS3ObjectFileSize` chama `prvGetS3ObjectFileSize` para recuperar o tamanho do objeto no bucket do S3. Atualmente, o servidor HTTP do Amazon S3 é compatível com solicitações HEAD usando uma URL pré-assinada, portanto, o 0 (zero) byte é solicitado. O campo de cabeçalho da resposta `Content-Range` contém o tamanho do arquivo. Uma resposta `206 Partial Content` é esperada do servidor. Qualquer outro código de status de resposta é um erro.

O código-fonte do `prvGetS3ObjectFileSize()` pode ser encontrado no GitHubsite.

Demonstração de download de coreHTTP básica do S3

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

Esta demonstração mostra como usar solicitações de intervalo para fazer download de arquivos do servidor HTTP do Amazon S3. As solicitações de intervalo têm suporte nativo na API do coreHTTP quando você usa `HTTPClient_AddRangeHeader` para criar a solicitação HTTP. As solicitações de intervalo são altamente recomendadas para um ambiente de microcontrolador. Ao baixar um arquivo grande em intervalos separados e não em uma única solicitação, cada seção do arquivo pode ser processada sem bloquear o soquete da rede. As solicitações de intervalo diminuem o risco de perda de pacotes, que exigem retransmissões na conexão TCP e, portanto, melhoram o consumo de energia do dispositivo.

Este exemplo usa uma interface de transporte de rede que usa mbedTLS para estabelecer uma conexão mutuamente autenticada entre um cliente de dispositivo de IoT executando coreHTTP e o servidor HTTP do Amazon S3.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Thread único versus thread múltiplo

Existem dois modelos de uso da coreHTTP, com thread único e com threads múltiplos (multitarefas). Embora a demonstração nesta seção execute a biblioteca HTTP em um thread, ela realmente demonstra como usar a coreHTTP em um ambiente com thread único (somente uma tarefa usa a API HTTP na demonstração). Embora as aplicações com thread único devam chamar repetidamente a biblioteca HTTP, as aplicações com threads múltiplos podem enviar solicitações HTTP em segundo plano em uma tarefa de agente (ou daemon).

Organização de código-fonte

O projeto de demonstração tem um nome `http_demo_s3_download.c` e pode ser encontrado no *`freertos`*`/demos/coreHTTP/` diretório e no GitHubsite.

Configuração de conexão do servidor HTTP do Amazon S3

Esta demonstração usa uma URL pré-assinada para se conectar ao servidor HTTP do Amazon S3 e autorizar o acesso ao objeto para download. A conexão TLS do servidor HTTP do Amazon S3 usa

somente a autenticação do servidor. No nível da aplicação, o acesso ao objeto é autenticado com os parâmetros na consulta de URL pré-assinada. Siga as etapas abaixo para configurar sua conexão com a AWS.

1. Configure uma AWS conta:

    a. Se ainda não o fez, crie e ative uma AWS conta.

    b. As contas e permissões são definidas usando AWS Identity and Access Management (IAM). O IAM permite gerenciar as permissões para cada usuário em sua conta. Por padrão, um usuário não tem permissões até que sejam concedidas pelo proprietário raiz.

        i. Para adicionar um usuário à sua AWS conta, consulte o Guia do usuário do IAM.

        ii. Conceda permissão à sua AWS conta para acessar os FreeRTOS adicionando estas AWS IoT políticas:

        - Amazon S3 FullAccess

2. Criar um bucket no S3 seguindo as etapas em Como criar um bucket do S3? no console Guia do usuário do Amazon Simple Storage Service.
3. Faça upload de um arquivo no S3 seguindo as etapas em Como fazer upload de arquivos e pastas em um bucket do S3?.
4. Gere um URL pré-assinado usando o script localizado em `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`. Para obter instruções de uso, consulte `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`.

Funcionalidade

A demonstração recupera primeiro o tamanho do arquivo. Depois, solicita cada intervalo de bytes sequencialmente, em um loop, com tamanhos de intervalo de `democonfigRANGE_REQUEST_LENGTH`.

O código-fonte da demonstração pode ser encontrado no GitHubsite.

Conexão ao servidor HTTP do Amazon S3

A função connectToServerWithBackoffRetries () tenta fazer uma conexão TCP com o servidor HTTP. Se a conexão falhar, ela tentará novamente após um tempo limite. O valor do tempo limite aumentará exponencialmente até que o número máximo de tentativas ou o valor do tempo limite seja

atingido. `connectToServerWithBackoffRetries()` retornará um status de falha se a conexão TCP com o servidor não puder ser estabelecida após o número configurado de tentativas.

A função `prvConnectToServer()` demonstra como estabelecer uma conexão com o servidor HTTP do Amazon S3 usando somente a autenticação do servidor. Ela usa a interface de transporte baseada em mbedTLS que é implementada no arquivo FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c.

O código-fonte de `prvConnectToServer()` pode ser encontrado em GitHub.

Criação de solicitação de intervalo

A função `HTTPClient_AddRangeHeader()` da API oferece suporte a serialização de um intervalo de bytes nos cabeçalhos da solicitação HTTP para formar uma solicitação de intervalo. As solicitações de intervalo são usadas nesta demonstração para recuperar o tamanho do arquivo e solicitar cada seção do arquivo.

A função `prvGetS3ObjectFileSize()` recupera o tamanho do arquivo no bucket do S3. O cabeçalho `Connection: keep-alive` é adicionado nessa primeira solicitação ao Amazon S3 para manter a conexão aberta após o envio da resposta. Atualmente, o servidor HTTP do S3 não oferece suporte às solicitações HEAD usando uma URL pré-assinada, portanto, o 0 (zero) byte é solicitado. O campo de cabeçalho da resposta `Content-Range` contém o tamanho do arquivo. Uma resposta `206 Partial Content` do servidor é esperada; qualquer outro código de status de resposta recebido é um erro.

O código-fonte de `prvGetS3ObjectFileSize()` pode ser encontrado em GitHub.

Depois de recuperar o tamanho do arquivo, essa demonstração cria uma nova solicitação de intervalo para cada intervalo de bytes do arquivo a ser baixado. Ela usa `HTTPClient_AddRangeHeader()` para cada seção do arquivo.

Envio de solicitações e recebimento de respostas

A função `prvDownloadS3ObjectFile()` envia as solicitações de intervalo em um loop até que o arquivo inteiro seja baixado. A função `HTTPClient_Send()` da API envia uma solicitação e recebe a resposta de forma síncrona. Quando a função retorna, a resposta é recebida em um `xResponse`. Em seguida, o código de status é verificado como `206 Partial Content` e o número de bytes baixados até o momento é incrementado pelo valor do cabeçalho `Content-Length`.

O código-fonte de `prvDownloadS3ObjectFile()` pode ser encontrado em GitHub.

Demonstração de threads múltiplas básica da coreHTTP

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

Esta demonstração usa as filas thread-safe do FreeRTOS para manter solicitações e respostas em espera para serem processadas. Nesta demonstração, tome nota de três tarefas.

- A tarefa principal espera que as solicitações apareçam na fila de solicitações. Ela enviará essas solicitações pela rede e, em seguida, colocará a resposta na fila de respostas.
- Uma tarefa de solicitação cria objetos de solicitação da biblioteca HTTP para enviar ao servidor e os coloca na fila de solicitações. Cada objeto de solicitação especifica um intervalo de bytes do arquivo S3 que a aplicação configurou para baixar.
- Uma tarefa de resposta espera as respostas aparecerem na fila de respostas. Ela registra todas as respostas que recebe.

Essa demonstração de threads múltiplos básica está configurada para usar uma conexão TLS somente com autenticação de servidor; isso é exigido pelo servidor HTTP do Amazon S3. A autenticação da camada da aplicação é feita usando os parâmetros Signature Version 4 na consulta de URL pré-assinada.

Organização de código-fonte

O projeto de demonstração tem um nome `http_demo_s3_download_multithreaded.c` e pode ser encontrado no *`freertos`*`/demos/coreHTTP/` diretório e no GitHubsite.

Compilação do projeto de demonstração

O projeto de demonstração usa a edição gratuita da comunidade do Visual Studio. Como criar a demonstração:

1. Abra o arquivo da solução do Visual Studio `mqtt_multitask_demo.sln` a partir do IDE do Visual Studio.
2. Selecione Criar solução no menu Compilar do IDE.

> **Note**
>
> Se você estiver usando o Microsoft Visual Studio 2017 ou anterior, deverá selecionar um conjunto de ferramentas de plataforma compatível com sua versão: Projeto -> Propriedades do RTOSDemos -> Conjunto de ferramentas da plataforma.

Configuração do projeto de demonstração

A demonstração usa a pilha TCP/IP FreeRTOS+TCP, portanto siga as instruções fornecidas para o projeto inicial de TCP/IP para:

1. Instale os componentes pré-requisitos (como o WinPcap).
2. Opcionalmente, defina um endereço IP estático ou dinâmico, endereço de gateway e máscara de rede.
3. Opcionalmente, defina um endereço MAC.
4. Selecione uma interface de rede Ethernet em sua máquina host.
5. É importante testar a conexão de rede antes de tentar executar a demonstração HTTP.

Configuração de conexão do servidor HTTP do Amazon S3

Siga as instruções para Configuração de conexão do servidor HTTP do Amazon S3 na demonstração básica de download da coreHTTP.

Funcionalidade

A demonstração cria um total de três tarefas:

- Uma que envia solicitações e recebe respostas pela rede.
- Uma que cria solicitações para enviar.
- Uma que processa as respostas recebidas.

Nesta demonstração, a tarefa principal:

1. Cria as filas de solicitação e resposta.
2. Cria a conexão com o servidor.
3. Cria as tarefas de solicitação e resposta.
4. Espera que a fila de solicitações envie solicitações pela rede.
5. Coloca as respostas recebidas pela rede na fila de respostas.

A tarefa de solicitação:

1. Cria cada uma das solicitações de intervalo.

A tarefa de resposta:

1. Processa cada uma das respostas recebidas.

Typedefs

A demonstração define as seguintes estruturas para oferecer suporte a threads múltiplos.

Itens de solicitação

As estruturas a seguir definem um item de solicitação a ser colocado na fila de solicitações. O item de solicitação é copiado para a fila depois que a tarefa de solicitação cria uma solicitação 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;
```

Item de resposta

As estruturas a seguir definem um item de resposta a ser colocado na fila de respostas. O item de resposta é copiado para a fila depois que a tarefa HTTP principal recebe uma resposta pela rede.

```
/**
 * @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;
```

Tarefa de envio HTTP principal

A tarefa principal da aplicação:

1. Analisa a URL pré-assinada do endereço do host para estabelecer uma conexão com o servidor HTTP do Amazon S3.
2. Analisa o URL pré-assinado do caminho para os objetos no bucket do S3.
3. Se conecta ao servidor HTTP do Amazon S3 usando TLS com a autenticação do servidor.
4. Cria as filas de solicitação e resposta.
5. Cria as tarefas de solicitação e resposta.

A função `prvHTTPDemoTask()` faz essa configuração e fornece o status da demonstração. O código-fonte para essa função pode ser encontrado no Github.

Na função `prvDownloadLoop()`, a tarefa principal bloqueia e aguarda as solicitações da fila de solicitações. Ao receber uma solicitação, ela a envia usando a função `HTTPClient_Send()` da API. Se a função da API obtiver êxito, ela colocará a resposta na fila de respostas.

O código-fonte para `prvDownloadLoop()` pode ser encontrado no GitHub.

Tarefa de solicitação HTTP

A tarefa de solicitação é especificada na função `prvRequestTask`. O código-fonte para essa função pode ser encontrado no Github.

A tarefa de solicitação recupera o tamanho do arquivo no bucket do Amazon S3. Isso é feito na função `prvGetS3ObjectFileSize`. O cabeçalho "Connection: keep-alive" é adicionado nessa solicitação ao Amazon S3 para manter a conexão aberta após o envio da resposta. Atualmente, o servidor HTTP do Amazon S3 não oferece suporte às solicitações HEAD usando uma URL pré-assinada, portanto, o 0 (zero) byte é solicitado. O campo de cabeçalho da resposta `Content-Range` contém o tamanho do arquivo. Uma resposta `206 Partial Content` do servidor é esperada; qualquer outro código de status de resposta recebido é um erro.

O código-fonte para `prvGetS3ObjectFileSize` pode ser encontrado no GitHub.

Depois de recuperar o tamanho do arquivo, a tarefa de solicitação continua solicitando cada intervalo do arquivo. Toda solicitação de intervalo é colocada na fila de solicitações para da tarefa principal enviar. Os intervalos de arquivos são configurados pelo usuário da demonstração na macro `democonfigRANGE_REQUEST_LENGTH`. As solicitações de intervalo têm suporte nativo na API da biblioteca do cliente HTTP usando a função `HTTPClient_AddRangeHeader`. A função `prvRequestS3ObjectRange` demonstra como usar `HTTPClient_AddRangeHeader()`.

O código-fonte para a função `prvRequestS3ObjectRange` pode ser encontrado no Github.

Tarefa de resposta HTTP

As tarefas de resposta aguardam na fila de respostas as respostas recebidas pela rede. A tarefa principal preenche a fila de respostas quando recebe uma resposta HTTP com êxito. Essa tarefa processa as respostas registrando o código de status, os cabeçalhos e o corpo. Uma aplicação em ambiente real, por exemplo, pode processar a resposta gravando o corpo da resposta na memória flash. Se o código de status da resposta não for `206 partial content`, a tarefa notificará a tarefa principal de que a demonstração deve obter falha. A tarefa da resposta é especificada na função `prvResponseTask`. O código-fonte para essa função pode ser encontrado no Github.

Demonstração da biblioteca Trabalhos do AWS IoT

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto

FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

A demonstração da biblioteca Trabalhos do AWS IoT mostra como se conectar ao serviço Trabalhos do AWS IoT por meio de uma conexão MQTT, recuperar um trabalho do AWS IoT e processá-lo em um dispositivo. O projeto de demonstração de Trabalhos do AWS IoT usa a porta do FreeRTOS para Windows, para que possa ser compilado e avaliado com a versão do Visual Studio Community no Windows. Nenhum hardware de microcontrolador é necessário. A demonstração estabelece uma conexão segura com o agente MQTT do AWS IoT usando TLS da mesma forma que Demonstração de autenticação mútua da coreMQTT.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Organização de código-fonte

O código de demonstração está no arquivo `jobs_demo.c` e pode ser encontrado no site do GitHub ou no diretório *`freertos`*`/demos/jobs_for_aws/`.

Configuração da conexão do agente MQTT do AWS IoT

Nesta demonstração, você usa uma conexão MQTT com o agente MQTT do AWS IoT. Essa conexão é configurada da mesma forma que Demonstração de autenticação mútua da coreMQTT.

Funcionalidade

A demonstração mostra o fluxo de trabalho usado para receber trabalhos do AWS IoT e processá-los em um dispositivo. A demonstração é interativa e exige que você crie trabalhos usando o console do AWS IoT ou o AWS Command Line Interface (AWS CLI). Para obter mais informações sobre como criar um trabalho, consulte create-job na Referência de comandos da AWS CLI. A demonstração exige que o documento do trabalho tenha uma chave `action`definida como `print` para exibir uma mensagem no console.

Veja o formato a seguir para esse documento de trabalho.

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

Você pode usar a AWS CLI para criar um trabalho como no exemplo de comando a seguir.

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

A demonstração também usa um documento de trabalho que tem a chave `action` definida `publish` para republicar a mensagem em um tópico. Veja o formato a seguir para esse documento de trabalho.

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

A demonstração continua até receber um documento de trabalho com a chave `action` configurada `exit` para sair da demonstração. O formato do documento de trabalho é o seguinte.

```
{
    "action: "exit"
}
```

Ponto de entrada da demonstração de trabalhos

O código-fonte para a função de ponto de entrada da demonstração de trabalhos pode ser encontrado no GitHub. Essa função executa as seguintes operações:

1. Estabeleça uma conexão MQTT usando as funções auxiliares em `mqtt_demo_helpers.c`.
2. Assine o tópico MQTT para a API `NextJobExecutionChanged`, usando funções auxiliares em `mqtt_demo_helpers.c`. A string do tópico é montada antecipadamente, usando macros definidas pela biblioteca Trabalhos do AWS IoT.

3. Publique no tópico MQTT para a API `StartNextPendingJobExecution`, usando funções auxiliares em `mqtt_demo_helpers.c`. A string do tópico é montada antecipadamente, usando macros definidas pela biblioteca Trabalhos do AWS IoT.
4. Chame repetidamente `MQTT_ProcessLoop` para receber mensagens de entrada que são entregues ao `prvEventCallback` para processamento.
5. Depois que a demonstração receber a ação de saída, cancele sua inscrição do tópico MQTT e desconecte-se usando as funções auxiliares no arquivo `mqtt_demo_helpers.c`.

Retorno de chamada para mensagens MQTT recebidas

A função `prvEventCallback` chama `Jobs_MatchTopic` da biblioteca Trabalhos do AWS IoT para classificar a mensagem MQTT de entrada. Se o tipo de mensagem corresponder a um novo trabalho, `prvNextJobHandler()` será chamado.

A função `prvNextJobHandler` e as funções que ela chama analisam o documento do trabalho a partir da mensagem formatada em JSON e executam a ação especificada pelo trabalho. A função `prvSendUpdateForJob` é particularmente interessante.

Enviar uma atualização para um trabalho em execução

A função `prvSendUpdateForJob ()` chama `Jobs_Update()` da biblioteca Jobs para preencher a string de tópico usada na operação de publicação do MQTT que se segue imediatamente.

Demonstrações do coreMQTT

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Essas demonstrações podem ajudar a saber como usar a biblioteca coreMQTT.

Tópicos

- Demonstração de autenticação mútua da coreMQTT

- Demonstração de compartilhamento de conexão da coreMQTT Agent

Demonstração de autenticação mútua da coreMQTT

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

O projeto de demonstração da coreMQTT mostra como estabelecer uma conexão com um agente MQTT usando TLS com autenticação mútua entre o cliente e o servidor. Esta demonstração usa uma implementação de interface de transporte baseada em mbedTLS para estabelecer uma conexão TLS autenticada pelo servidor e pelo cliente e demonstra um fluxo de trabalho de publicação e assinatura de MQTT no nível de QoS 1. Ela assina um filtro de tópicos, em seguida, publica em tópicos que correspondem ao filtro e aguarda o recebimento dessas mensagens do servidor no nível de QoS 1. Esse ciclo de publicar para o agente e receber de volta a mesma mensagem do agente se repete indefinidamente. As mensagens nesta demonstração são enviadas QoS 1, o que garante pelo menos uma entrega de acordo com a especificação MQTT.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Código-fonte

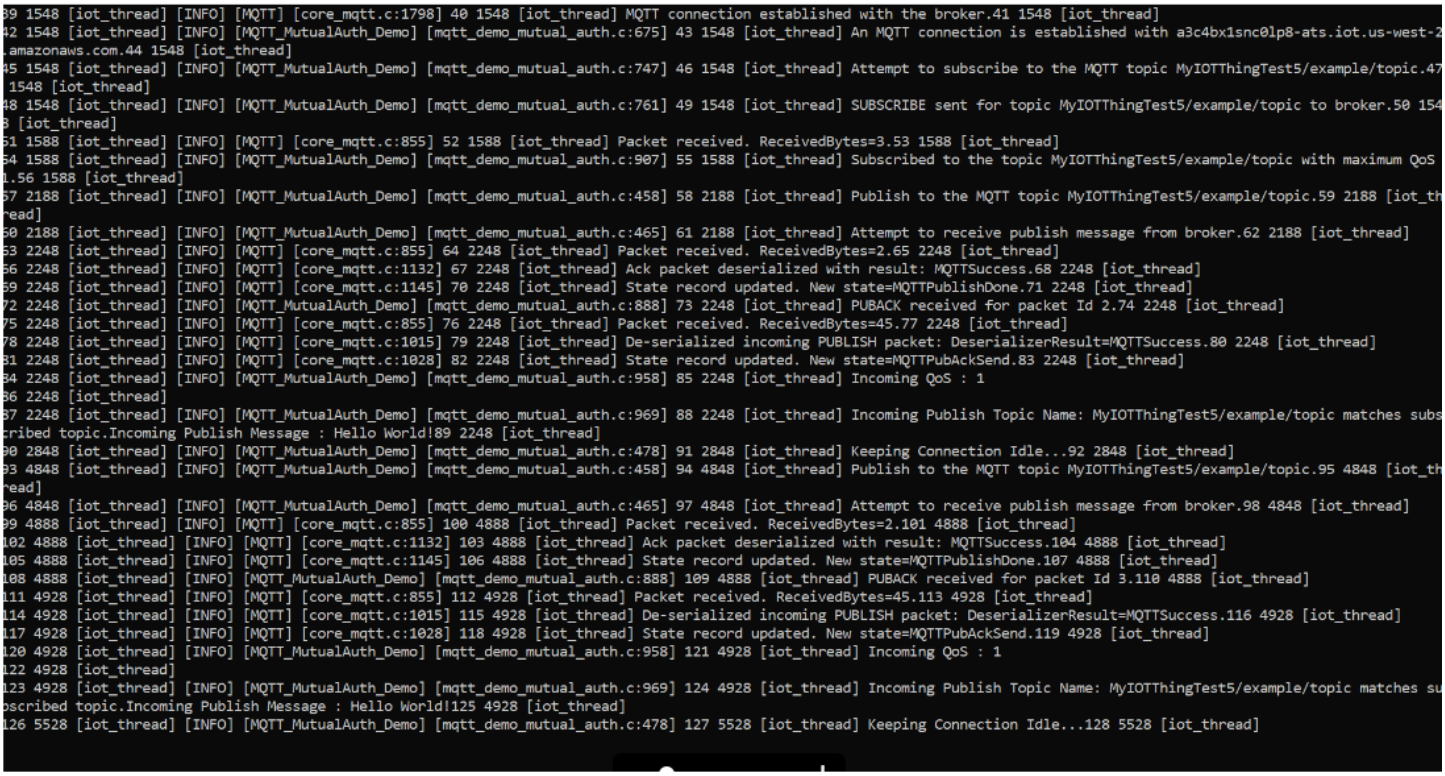
O arquivo de demonstração de origem é chamado `mqtt_demo_mutual_auth.c` e pode ser encontrado no diretório *`freertos`*`/demos/coreMQTT/` e no site do GitHub.

Funcionalidade

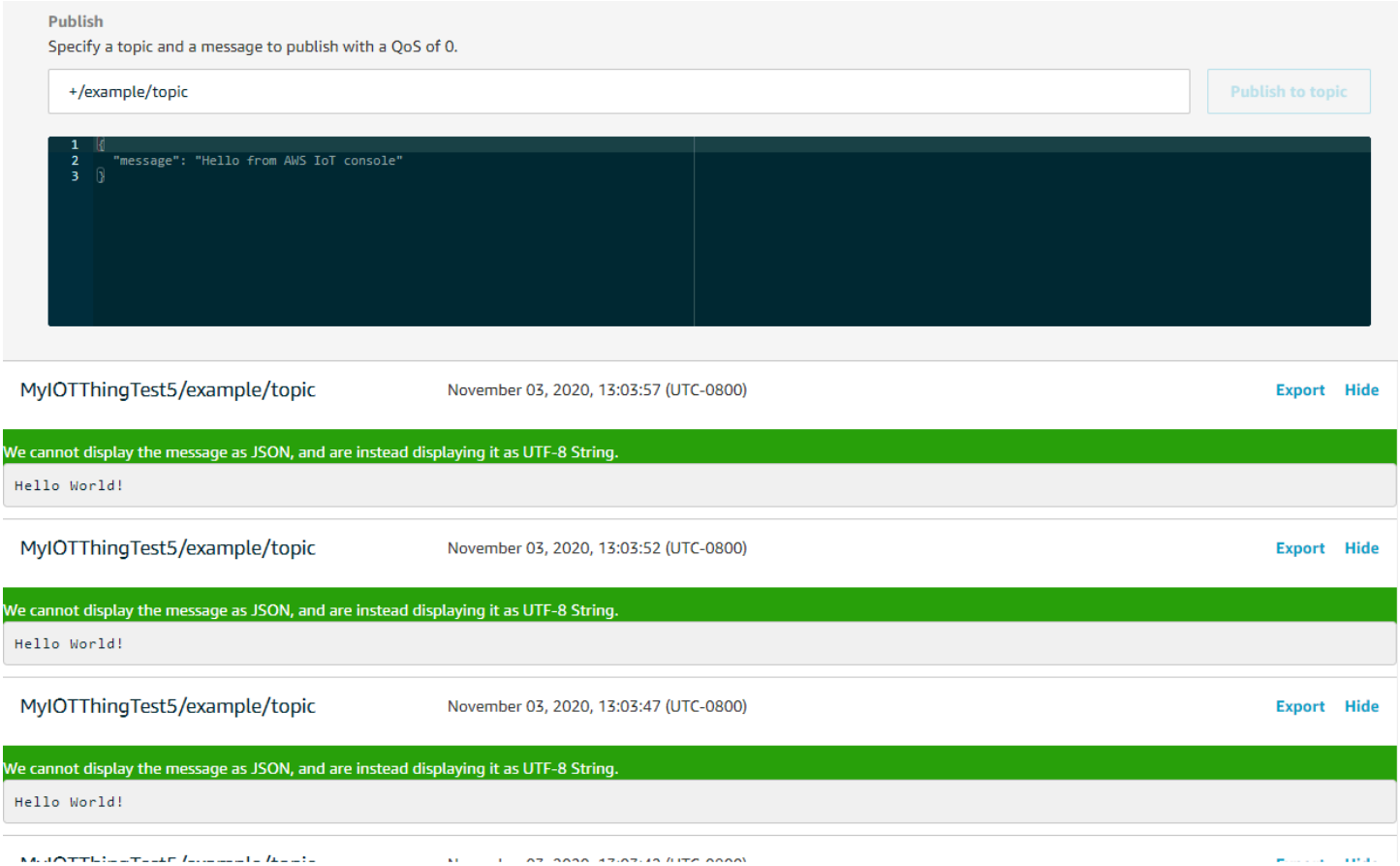
A demonstração cria uma única tarefa de aplicativo que percorre um conjunto de exemplos demonstrando como se conectar ao agente, se inscrever em um tópico no agente, publicar em um

tópico no agente e, enfim, se desconectar do agente. O aplicativo de demonstração assina e publica o mesmo tópico. Cada vez que a demonstração publica uma mensagem para o agente MQTT, o agente envia a mesma mensagem de volta para o aplicativo de demonstração.

A conclusão com êxito da demonstração gerará um resultado semelhante ao da imagem a seguir.



O console AWS IoT gerará uma saída semelhante a da imagem a seguir.



Lógica de repetição com recuo exponencial e tremulação

A função prvBackoffForRetry mostra como operações de rede com falha no servidor, por exemplo, conexões TLS ou solicitações de assinatura de MQTT, podem ser repetidas com recuo exponencial e tremulação. A função calcula o período de recuo para a próxima tentativa de repetição e executa o recuo se as tentativas de repetição não forem esgotadas. Como o cálculo do período de recuo exige a geração de um número aleatório, a função usa o módulo PKCS11 para gerar o número aleatório. O uso do módulo PKCS11 permite o acesso a um TRNG (Gerador de Números Aleatórios Verdadeiros) se a plataforma do fornecedor oferecer suporte a isso. Recomendamos que você alimente o gerador de números aleatórios com uma fonte de entropia específica do dispositivo para reduzir a probabilidade de colisões de dispositivos durante as novas tentativas de conexão.

Conexão ao agente MQTT

A função prvConnectToServerWithBackoffRetries tenta fazer uma conexão TLS mutuamente autenticada com o agente MQTT. Se a conexão falhar, ela tentará novamente após um período de recuo. O período de recuo aumentará exponencialmente até que o número máximo de tentativas ou o valor do período de recuo seja atingido. A função

`BackoffAlgorithm_GetNextBackoff` fornecerá um valor de recuo aumentando exponencialmente e retornará `RetryUtilsRetriesExhausted` quando o número máximo de tentativas for atingido. A função `prvConnectToServerWithBackoffRetries` retornará um status de falha se a conexão TLS com o agente não puder ser estabelecida após o número configurado de tentativas.

A função prvCreateMqttConnectionWithBroker demonstra como estabelecer uma conexão MQTT com um agente MQTT com uma sessão limpa. Ela usa a interface de transporte TLS, que é implementada no arquivo `FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c`. Lembre-se de que estamos configurando os segundos de keep-alive para o agente em `xConnectInfo`.

A próxima função mostra como a interface de transporte TLS e a função de tempo são definidas em um contexto MQTT usando a função `MQTT_Init`. E mostra também como um ponteiro da função de retorno de chamada de evento (`prvEventCallback`) é definido. Esse retorno de chamada é usado para relatar mensagens recebidas.

Assinatura em um tópico MQTT

A função prvMQTTSubscribeWithBackoffRetries demonstra como se inscrever em um filtro de tópicos no agente MQTT. O exemplo demonstra como assinar um filtro de tópico, mas é possível passar uma lista de filtros de tópicos na mesma chamada de API de assinatura para assinar mais de um filtro de tópico. Além disso, caso o agente MQTT rejeite a solicitação de assinatura, a assinatura tentará novamente, com recuo exponencial, por `RETRY_MAX_ATTEMPTS`.

Publicar em um tópico

A função prvMQTTPublishToTopic demonstra como publicar em um tópico no agente MQTT.

Recebimento de mensagens

O aplicativo registra uma função de retorno de chamada de evento antes de se conectar ao agente, conforme descrito anteriormente. A função `prvMQTTDemoTask` chama a função `MQTT_ProcessLoop` para receber mensagens. Quando uma mensagem MQTT é recebida, ela chama a função de retorno de chamada do evento registrada pelo aplicativo. A função prvEventCallback é um exemplo dessa função de retorno de chamada de evento. `prvEventCallback` examina o tipo de pacote recebido e chama o manipulador apropriado. No exemplo abaixo, a função chama `prvMQTTProcessIncomingPublish()` para manipular mensagens de publicação recebidas ou `prvMQTTProcessResponse()` para manipular confirmações (ACK).

Processamento de pacotes de publicação MQTT recebidos

A função prvMQTTProcessIncomingPublish demonstra como processar um pacote de publicação do agente MQTT.

Cancelar a inscrição em um tópico

A última etapa do fluxo de trabalho é cancelar a assinatura do tópico para que o agente não envie mensagens publicadas do `mqttexampleTOPIC`. Confira aqui a definição da função prvMQTTUnsubscribeFromTopic.

Alteração da CA raiz usada na demonstração

Por padrão, as demonstrações do FreeRTOS usam o certificado o Amazon Root CA 1 (chave RSA de 2048 bits) para obter autenticação no servidor do AWS IoT Core. É possível usar outros certificados CA para autenticação do servidor, incluindo o certificado Amazon Root CA 3 (chave ECC de 256 bits). Como alterar a CA raiz da demonstração de autenticação mútua da coreMQTT:

1. Em um editor de texto, abra o arquivo *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/mqtt_demo_mutual_auth_config.h`.
2. No arquivo, localize a linha a seguir.

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   Remover o comentário dessa linha e, se necessário, movê-lo para depois do final `*/` do bloco de comentários.
3. Copie o certificado CA que você deseja usar e cole-o no texto `"...insert here..."`. O resultado será algo semelhante a este exemplo:

   ```
   #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) Você pode alterar a CA raiz de outras demonstrações. Repita as etapas de 1 a 3 para cada arquivo *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h.

Demonstração de compartilhamento de conexão da coreMQTT Agent

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

O projeto de demonstração de compartilhamento de conexão da coreMQTT mostra como usar um aplicativo com threads múltiplas para estabelecer uma conexão com um agente MQTT da AWS usando TLS com autenticação mútua entre o cliente e o servidor. Esta demonstração usa uma implementação de interface de transporte baseada em mbedTLS para estabelecer uma conexão TLS autenticada pelo servidor e pelo cliente e demonstra um fluxo de trabalho de publicação e assinatura de MQTT no nível de QoS 1. A demonstração se inscreve em um filtro de tópicos, publica em tópicos que correspondem ao filtro e, em seguida, aguarda para receber essas mensagens do servidor no nível de QoS 1. Esse ciclo de publicar para o agente e receber de volta a mesma mensagem do agente se repete indefinidamente. As mensagens nesta demonstração são enviadas QoS 1, o que garante pelo menos uma entrega de acordo com a especificação MQTT.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Esta demonstração usa uma fila de threads segura para manter comandos para interagir com a API MQTT. Tome nota de duas tarefas nesta demonstração.

- Uma tarefa da coreMQTT Agent (principal) processa os comandos da fila de comandos enquanto outras tarefas os enfileiram. Essa tarefa entra em um loop, durante o qual processa os comandos da fila de comandos. Se um comando de encerramento for recebido, essa tarefa escapará do loop.
- Uma tarefa de subpub de demonstração cria uma assinatura para um tópico do MQTT, depois cria operações de publicação e as envia para a fila de comandos. Essas operações de publicação são então executadas pela tarefa da coreMQTT Agent A tarefa de subpub de demonstração aguarda a conclusão da publicação, indicada pela execução do retorno de chamada de conclusão do comando, depois insere um pequeno atraso antes de iniciar a próxima publicação. Essa tarefa mostra exemplos de como as tarefas do aplicativo usariam a API da coreMQTT Agent.

A coreMQTT Agent invoca uma única função de retorno de chamada para mensagens de publicação recebidas. Essa demonstração também inclui um gerenciador de assinaturas que permite que as tarefas especifiquem um retorno de chamada a ser invocado para receber mensagens de publicação nos tópicos que assinaram. A chamada de retorno de publicação recebida do agente nesta demonstração invoca o gerenciador de assinaturas para distribuir publicações para toda tarefa que tenha registrado uma assinatura.

Esta demonstração usa uma conexão TLS com autenticação mútua para se conectar a AWS. Se a rede for desconectada inesperadamente durante a demonstração, o cliente tentará se reconectar usando a lógica de recuo exponencial. Se o cliente se reconectar com êxito, mas o agente não conseguir retomar a sessão anterior, o cliente assinará novamente os mesmos tópicos da sessão anterior.

Thread única versus thread múltipla

Existem dois modelos de uso da coreMQTT, com thread único e com thread múltiplo (multitarefas). O modelo com thread único usa a biblioteca coreMQTT somente de um thread e exige que você faça chamadas explícitas repetidas na biblioteca MQTT. Já os casos de uso com thread múltiplo podem executar o protocolo MQTT em segundo plano em uma tarefa de agente (ou daemon), conforme mostrado na demonstração documentada aqui. Ao executar o protocolo MQTT em uma tarefa do agente, você não precisa gerenciar explicitamente nenhum estado do MQTT nem chamar a função da API `MQTT_ProcessLoop`. Além disso, quando você usa uma tarefa de agente, várias tarefas do aplicativo podem compartilhar uma única conexão MQTT sem a necessidade de primitivos de sincronização, como mutexes.

Código-fonte

O arquivo de demonstração de origem são chamados `mqtt_agent_task.c` e `simple_sub_pub_demo.c` e podem ser encontrados no diretório *`freertos`*`/demos/coreMQTT_Agent/` e no site do GitHub.

Funcionalidade

Esta demonstração cria pelo menos duas tarefas: uma primária que processa solicitações de chamadas da API MQTT e um número configurável de subtarefas que criam essas solicitações. Nesta demonstração, a tarefa principal cria as subtarefas, chama o ciclo de processamento e faz a limpeza depois. A tarefa principal cria uma única conexão MQTT com o agente que é compartilhada entre as subtarefas. As subtarefas criam uma assinatura MQTT no agente e depois publicam mensagens nele. Cada subtarefa usa um tópico exclusivo para suas publicações.

Tarefa principal

A tarefa principal do aplicativo, RunCoreMQTTAgentDemo, estabelece uma sessão MQTT, cria as subtarefas e executa o loop de processamento MQTTAgent_CommandLoop até que um comando de encerramento seja recebido. Se a rede se desconectar inesperadamente, a demonstração se reconectará ao agente em segundo plano e restabelecerá as assinaturas no agente. Após o término do loop de processamento, ele se desconectará do corretor.

Comandos

Ao invocar uma API da coreMQTT Agent, ela cria um comando que é enviado para a fila de tarefas do agente, que é processado em `MQTTAgent_CommandLoop()`. No momento em que o comando é criado, parâmetros de contexto e retorno de chamada de conclusão opcionais podem ser passados. Assim que o comando correspondente é concluído, o retorno de chamada de conclusão será invocado com o contexto passado e todos os valores de retorno criados como resultado do comando. A assinatura para o retorno de chamada de conclusão é a seguinte:

```
typedef void (* MQTTAgentCommandCallback_t )( void * pCmdCallbackContext,
                                              MQTTAgentReturnInfo_t * pReturnInfo );
```

O contexto de conclusão do comando é definido pelo usuário; para esta demonstração, é: struct MQTTAgentCommandContext.

Os comandos são considerados concluídos quando:

- Assina, cancela a assinatura e publica com QoS > 0: depois que o pacote de confirmação correspondente for recebido.
- Todas as outras operações: depois que a API coreMQTT correspondente for invocada.

Toda estrutura usada pelo comando, incluindo informações de publicação, informações de assinatura e contextos de conclusão, deve permanecer no escopo até que o comando seja concluído. Uma tarefa de chamada não deve reutilizar nenhuma das estruturas de um comando antes da invocação do retorno de chamada de conclusão. Observe que, como o retorno de chamada de conclusão é invocado pelo agente MQTT, ele será executado com o contexto de thread da tarefa do agente, não com a tarefa que criou o comando. Mecanismos de comunicação entre processos, como notificações de tarefas ou filas, podem ser usados para sinalizar a tarefa de chamada da conclusão do comando.

Execução do loop de comando

Os comandos são processados continuamente em `MQTTAgent_CommandLoop()`.
Se não houver comandos a serem processados, o loop aguardará no máximo de `MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME` para que um seja adicionado à fila e, se nenhum comando for adicionado, executará uma iteração única de `MQTT_ProcessLoop()`. Isso garante que o MQTT Keep-Alive seja gerenciado e que todas as publicações recebidas sejam recebidas mesmo quando não existem comandos na fila.

A função de loop de comando retornará por um dos seguintes motivos:

- Além disso, um comando retorna qualquer código de status `MQTTSuccess`. O status do erro é retornado pelo loop de comando, então você pode decidir como lidar com isso. Nesta demonstração, a conexão TCP é restabelecida e uma tentativa de reconexão é feita. Se houver algum erro, uma reconexão pode ocorrer em segundo plano sem qualquer intervenção de outras tarefas usando o MQTT.
- Um comando de desconexão (de `MQTTAgent_Disconnect`) é processado. O loop de comando é encerrado para que o TCP possa ser desconectado.
- Um comando de encerramento (de `MQTTAgent_Terminate`) é processado. Esse comando também marca os comandos que ainda estão na fila ou aguardando um pacote de confirmação como um erro, com um código de retorno de `MQTTRecvFailed`.

Gerenciador de assinaturas

Como a demonstração usa vários tópicos, um gerenciador de assinaturas é uma maneira conveniente de associar tópicos assinados com retornos de chamada ou tarefas exclusivas. O gerenciador de assinaturas nesta demonstração tem thread único, portanto, não deve ser usado por várias tarefas ao mesmo tempo. Nesta demonstração, as funções do gerenciador de assinaturas são chamadas somente a partir das funções de retorno de chamada que são passadas para o agente MQTT e executadas somente com o contexto de thread da tarefa do agente.

Tarefa de assinatura/publicação simples

Cada instância do prvSimpleSubscribePublishTask cria uma assinatura para um tópico do MQTT e cria operações de publicação para esse tópico. Para demonstrar vários tipos de publicação, até tarefas com números pares usam QoS 0 (que são concluídas quando o pacote de publicação é enviado) e tarefas ímpares usam QoS 1 (que são concluídas após o recebimento de um pacote PUBACK).

Aplicativo de demonstração de atualizações remotas

O FreeRTOS inclui um aplicativo de demonstração que demonstra a funcionalidade da biblioteca sem fios (OTA). O aplicativo de demonstração OTA está localizado no arquivo.*freertos*/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c ou *freertos*/demos/ota/ota_demo_core_http/ota_demo_core_http.c.

O aplicativo de demonstração OTA faz o seguinte:

1. Inicializa a pilha de rede do FreeRTOS e o grupo de buffers MQTT.
2. Cria uma tarefa para exercer a biblioteca OTA usando `vRunOTAUpdateDemo()`.
3. Cria um cliente MQTT usando `_establishMqttConnection()`.
4. Se conecta ao agente MQTT de AWS IoT usando `IotMqtt_Connect()` e registra um retorno de chamada de desconexão do MQTT: `prvNetworkDisconnectCallback`.
5. Chama `OTA_AgentInit()` para criar a tarefa OTA e registra um retorno de chamada a ser usado quando a tarefa OTA é concluída.
6. Reutiliza a conexão MQTT com `xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. Se o MQTT se desconectar, o aplicativo suspenderá o agente OTA, tentará se reconectar usando atraso exponencial com tremulação e, em seguida, reiniciará o agente OTA.

Antes de usar as atualizações OTA, conclua todos os pré-requisitos no Atualizações sem fios do FreeRTOS

Após concluir a configuração para atualizações OTA, faça download, compile, instale e execute a demonstração OTA do FreeRTOS em uma plataforma que ofereça suporte ao recurso OTA. As instruções de demonstração específicas do dispositivo estão disponíveis para os seguintes dispositivos qualificados para o FreeRTOS:

- Texas Instruments CC3220SF-LAUNCHXL
- Microchip Curiosity PIC32MZEF
- Espressif ESP32
- Fazer download, compilação, atualização e execução de demonstração OTA do FreeRTOS no RX65N da Renesas

Após compilar, instalar e executar o aplicativo de demonstração OTA no seu dispositivo, você poderá usar o console da AWS IoT ou a AWS CLI para criar um trabalho de atualização OTA. Depois de criar uma tarefa de atualização OTA, conecte um emulador de terminal para ver o progresso da atualização OTA. Anote os erros gerados durante o processo.

Uma tarefa de atualização OTA bem-sucedida exibe um resultado semelhante ao seguinte. Algumas linhas neste exemplo foram removidas da listagem para facilitar.

```
    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.
```

Configurações de demonstração sem fios

As configurações de demonstração OTA são opções de configuração específicas de demonstração fornecidas em `aws_iot_ota_update_demo.c`. Essas configurações são diferentes das configurações da biblioteca OTA fornecidas no arquivo de configuração da biblioteca OTA.

OTA_DEMO_KEEP_ALIVE_SECONDS

Para o cliente MQTT, essa configuração é o intervalo máximo de tempo que pode decorrer entre o término da transmissão de um pacote de controle e o início do envio do próximo. Na falta de um pacote de controle, um PINGREQ é enviado. O agente deve desconectar um cliente que não envia uma mensagem ou um pacote PINGREQ de vez em quando durante esse intervalo keep-alive. Essa configuração deve ser ajustada com base nos requisitos do aplicativo.

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

O intervalo base, em segundos, antes de tentar a conexão de rede novamente. A demonstração OTA tentará se reconectar após esse intervalo de tempo base. O intervalo dobrará após cada tentativa com falha. Um atraso aleatório, até o máximo desse atraso de base, também é adicionado ao intervalo.

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

O intervalo máximo, em segundos, antes de tentar a conexão de rede novamente. O atraso de reconexão dobrará em cada tentativa com falha, mas só pode chegar até esse valor máximo, mais uma tremulação do mesmo intervalo.

Fazer download, compilação, instalação e execução de demonstração OTA do FreeRTOS no CC3220SF-LAUNCHXL da Texas Instruments

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Como fazer download do FreeRTOS e do código de demonstração OTA

- É possível baixar o código-fonte no site GitHub em https://github.com/FreeRTOS/FreeRTOS.

Para criar o aplicativo de demonstração

1. Siga as instruções em Conceitos básicos do FreeRTOS para importar o projeto `aws_demos` para o Code Composer Studio, configurar o endpoint da AWS IoT, o SSID e a senha do Wi-Fi, uma chave privada e um certificado para a 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` e defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Crie a solução e verifique se ela foi criada sem erros.
4. Inicie um emulador de terminal e use as seguintes configurações para se conectar à placa:
   - Taxa de baud: 115200
   - Bits de dados: 8
   - Paridade: nenhum
   - Bits de parada: 1
5. Execute o projeto na placa para confirmar se ele pode se conectar ao Wi-Fi e ao agente de mensagens MQTT da AWS IoT.

Quando executado, o emulador de terminal deve exibir um texto como o seguinte:

```
    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
```

Fazer download, compilação, instalação e execução de demonstração OTA do FreeRTOS no Microchip Curiosity PIC32MZEF

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

> **Note**
>
> De acordo com a Microchip, estamos removendo o Curiosity PIC32MZEF (DM320104) da ramificação principal do repositório de Integração de Referência do FreeRTOS e não o carregaremos mais em novas versões. A Microchip emitiu um aviso oficial de que o PIC32MZEF (DM320104) não é mais recomendado para novos designs. Os projetos e o código-fonte do PIC32MZEF ainda podem ser acessados por meio das tags de lançamento anteriores. A Microchip recomenda que os clientes usem a placa de desenvolvimento Curiosity PIC32MZ-EF-2.0 (DM320209) para novos designs. A plataforma PIC32MZv1 ainda pode ser encontrada na versão v202012.00 do repositório de Integração de Referência do FreeRTOS. No entanto, não há mais suporte para a plataforma não na versão v202107.00 da Referência do FreeRTOS.

Como fazer download do código de demonstração OTA do FreeRTOS

- É possível baixar o código-fonte no site GitHub em https://github.com/FreeRTOS/FreeRTOS.

Para criar o aplicativo de demonstração de atualização OTA

1. Siga as instruções em Conceitos básicos do FreeRTOS para importar o projeto `aws_demos` para o MPLAB X IDE, configurar o endpoint da AWS IoT, o SSID e a senha do Wi-Fi, uma chave privada e um certificado para a placa.
2. Abra o arquivo `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` e insira seu certificado.

```
[] = "your-certificate-key";
```

3. Cole o conteúdo do certificado de assinatura de código aqui:

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Siga o mesmo formato que `aws_clientcredential_keys.h`, cada linha deve terminar com o novo caractere de linha ('\n') e estar entre aspas.

Por exemplo, o certificado deve ser semelhante ao seguinte:

```
"-----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 o Python 3 ou posterior.
5. Instale pyOpenSSL executando `pip install pyopenssl`.
6. Copie o certificado de assinatura de código no formato .pem no caminho `demos/ota/bootloader/utility/codesigner_cert_utility/`. Renomeie o arquivo de certificado `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` e defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
8. Crie a solução e verifique se ela foi criada sem erros.
9. Inicie um emulador de terminal e use as seguintes configurações para se conectar à placa:
   - Taxa de baud: 115200
   - Bits de dados: 8

- Paridade: nenhum
- Bits de parada: 1

10. Desconecte o depurador da placa e execute o projeto na placa para confirmar se ele pode se conectar ao Wi-Fi e ao agente de mensagens MQTT da AWS IoT.

Quando você executa o projeto, o MPLAB X IDE deve abrir uma janela de saída. Certifique-se de que a guia ICD4 esteja selecionada. Você deve ver a saída a seguir.

```
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
```

O emulador de terminal deve exibir um texto como o seguinte:

```
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
```

Essa saída mostra que o Microchip Curiosity PIC32MZEF pode se conectar à AWS IoT e assinar os tópicos MQTT necessários para as atualizações OTA. As mensagens `Missing job parameter` são esperadas porque não há tarefas de atualização OTA pendentes.

Fazer download, compilação, instalação e execução de demonstração de OTA do FreeRTOS no Espressif ESP32

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

1. Faça o download da fonte do FreeRTOS no GitHub. Consulte o arquivo README.md para obter instruções. Crie um projeto no seu IDE que inclua todas as fontes e bibliotecas necessárias.
2. Siga as instruções em Conceitos básicos do Espressif para configurar a cadeia de ferramentas necessária baseada em GCC.
3. Abra *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` e defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
4. Crie o projeto de demonstração, executando `make` no diretório `vendors/espressif/boards/esp32/aws_demos`. Você pode fazer o flash do programa de demonstração e verificar sua saída executando `make flash monitor`, conforme descrito em Conceitos básicos do Espressif.
5. Antes de executar a demonstração da atualização OTA:

   - Abra *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` e defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

- Abra `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` e copie seu certificado de assinatura de código SHA-256/ECDSA em:

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Fazer download, compilação, atualização e execução de demonstração OTA do FreeRTOS no RX65N da Renesas

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Esse capítulo mostra como você faz download, compilação, atualização e execução das aplicações de demonstração OTA do FreeRTOS no RX65N da Renesas.

Tópicos

- Configuração do ambiente operacional
- Configure seus AWS recursos
- Importe, configure o arquivo de cabeçalho e compile aws_demos e boot_loader

Configuração do ambiente operacional

Os procedimentos nesta seção usam os seguintes ambientes:

- IDE: e^2 studio 7.8.0, e^2 studio 2020-07
- Cadeia de ferramentas: compilador CCRX 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

Como configurar o hardware

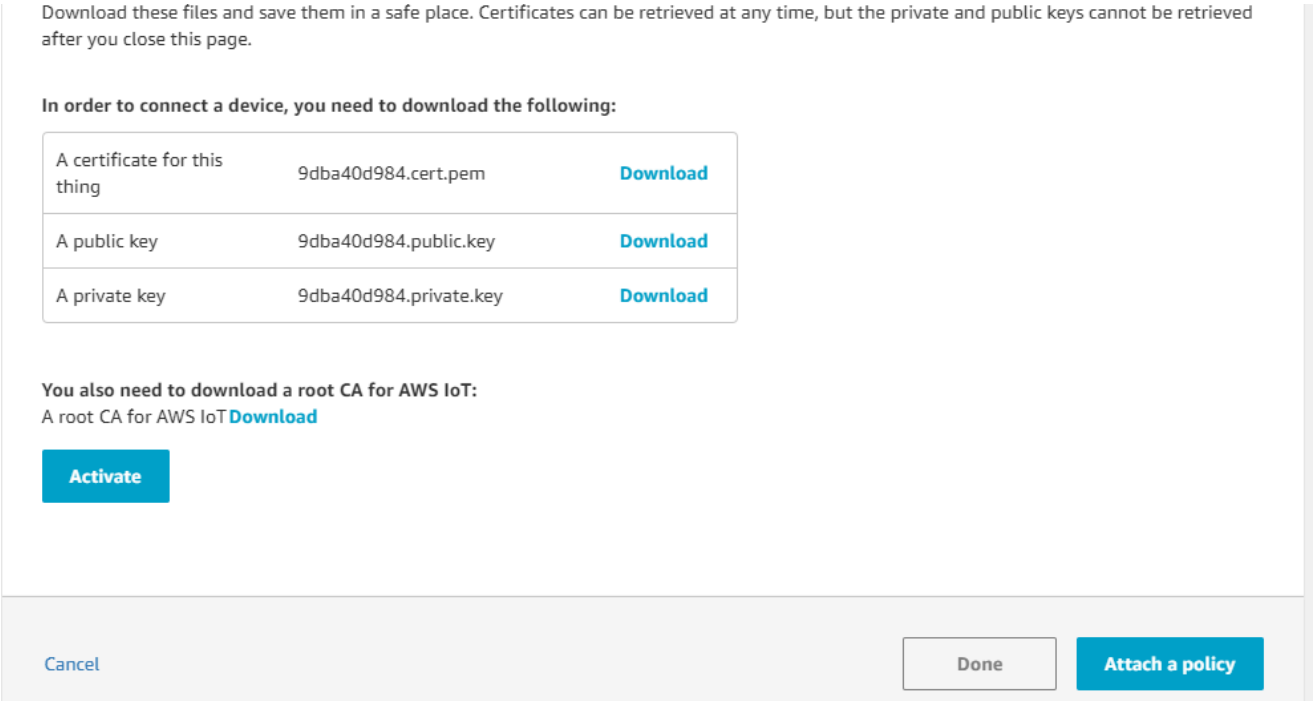
1. Conecte o emulador E^2 Lite e a porta serial USB à placa RX65N e ao PC.
2. Conecte a fonte de alimentação ao RX65N.

Configure seus AWS recursos

1. Para executar as demonstrações do FreeRTOS, você deve ter AWS uma conta com um usuário do IAM que tenha permissão para acessar os serviços. AWS IoT Caso ainda não tenha, siga as etapas em Configurando sua AWS conta e permissões.
2. Para configurar as atualizações OTA, siga as etapas em Pré-requisitos de atualização do OTA. Em especial, siga as etapas em Pré-requisitos para atualizações de OTA usando MQTT.
3. Abra o console de AWS IoT.
4. No painel de navegação à esquerda, escolha Gerenciar e, depois, Coisas para criar uma coisa.

   Uma coisa é uma representação de um dispositivo ou entidade lógica em AWS IoT. Ela pode ser um dispositivo físico ou sensor (por exemplo, uma lâmpada ou um interruptor em uma parede). Também pode ser uma entidade lógica, como uma instância de um aplicativo ou entidade física que não se conecta AWS IoT, mas está relacionada a dispositivos que o fazem (por exemplo, um carro com sensores de motor ou um painel de controle). AWS IoT fornece um registro de coisas que ajuda você a gerenciar suas coisas.

   a. Escolha Criar e Criar uma única coisa.
   b. Insira o Nome da coisa e escolha Próximo.
   c. Selecione Criar certificado.
   d. Faça download dos três arquivos criados e escolha Ativar.
   e. Selecione a opção Anexar uma política.



f. Selecione a política que você criou em Política de dispositivo.

Cada dispositivo que recebe uma atualização OTA usando o MQTT deve ser registrado como um item AWS IoT e deve ter uma política anexada, como a listada. Você pode encontrar mais informações sobre os itens nos objetos `"Resource"` e `"Action"` em Ações da política principal do AWS IoT e Recursos da ação principal do AWS IoT.

Observações

- As `iot:Connect` permissões permitem que seu dispositivo se conecte AWS IoT ao MQTT.
- As permissões de `iot:Subscribe` e `iot:Publish` para tópicos de trabalhos da AWS IoT (`.../jobs/*`) permitem que o dispositivo conectado receba notificações de trabalho e documentos de trabalho, e publique o estado de conclusão da execução de um trabalho.
- As `iot:Publish` permissões `iot:Subscribe` e sobre os tópicos de fluxos AWS IoT OTA (`.../streams/*`) permitem que o dispositivo conectado busque dados de atualização do OTA de. AWS IoT Essas permissões são necessárias para executar atualizações de firmware pelo MQTT.
- As `iot:Receive` permissões AWS IoT Core permitem publicar mensagens sobre esses tópicos no dispositivo conectado. Essa permissão é verificada em cada entrega de uma

mensagem de MQTT. Você pode usar essa permissão para revogar o acesso a clientes que estão inscritos em um tópico atualmente.

5. Para criar um perfil de assinatura de código e registrar um certificado de assinatura de código em. AWS

   a. Para criar as chaves e a certificação, consulte a seção 7.3 "Geração de pares de chaves ECDSA-SHA256 com OpenSSL" na Política de criação de atualização de firmware de MCU da Renesas.

   b. Abra o console de AWS IoT. No painel de navegação esquerdo, selecione Gerenciar, depois, Trabalhos. Selecione Criar um trabalho, depois Criar trabalho de atualização OTA.

   c. Em Selecionar dispositivos para atualizar, escolha Selecionar e escolha a coisa que você criou anteriormente. Escolha Próximo.

   d. Na página Criar um trabalho de atualização OTA do FreeRTOS, faça o seguinte:

      i. Em Selecionar o protocolo para transferência de imagem do firmware, escolha MQTT.

      ii. Em Selecionar e assinar a imagem de firmware, escolha Assinar uma nova imagem de firmware para mim.

      iii. Em Perfil de assinatura de código, escolha Criar.

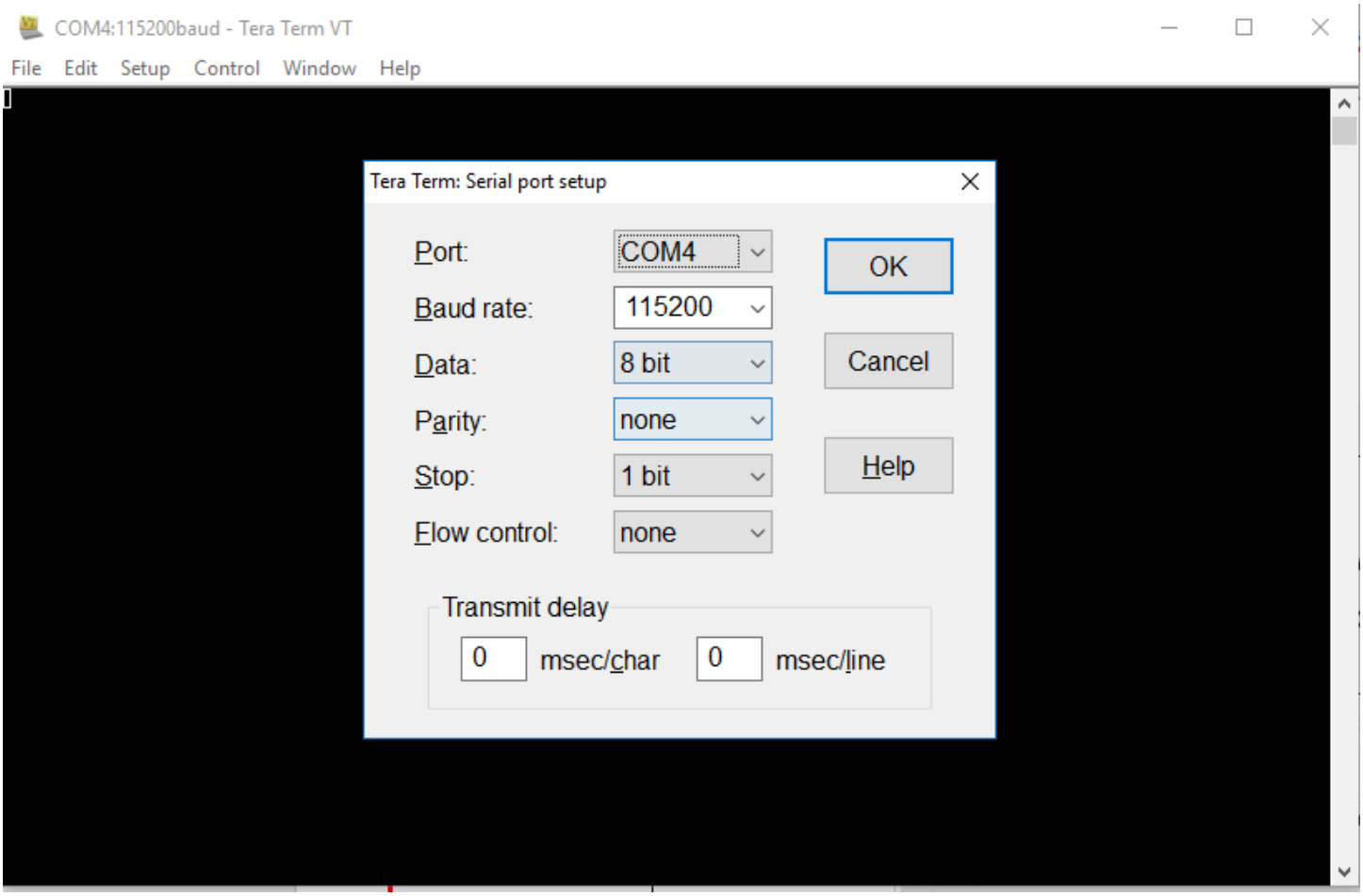
      iv. Na janela Criar um perfil de assinatura de código, insira um nome de perfil. Para a plataforma de hardware de dispositivo, selecione o Windows Simulator. Para o certificado de assinatura de código, escolha Importar.

      v. Navegue para selecionar o certificado (`secp256r1.crt`), a chave privada do certificado (`secp256r1.key`) e a cadeia de certificados (`ca.crt`).

      vi. Insira um nome de caminho do certificado de assinatura de código no dispositivo. Em seguida, selecione Criar.

6. Para conceder acesso à assinatura de código para AWS IoT, siga as etapas emConceder acesso ao Code Signing para AWS IoT.

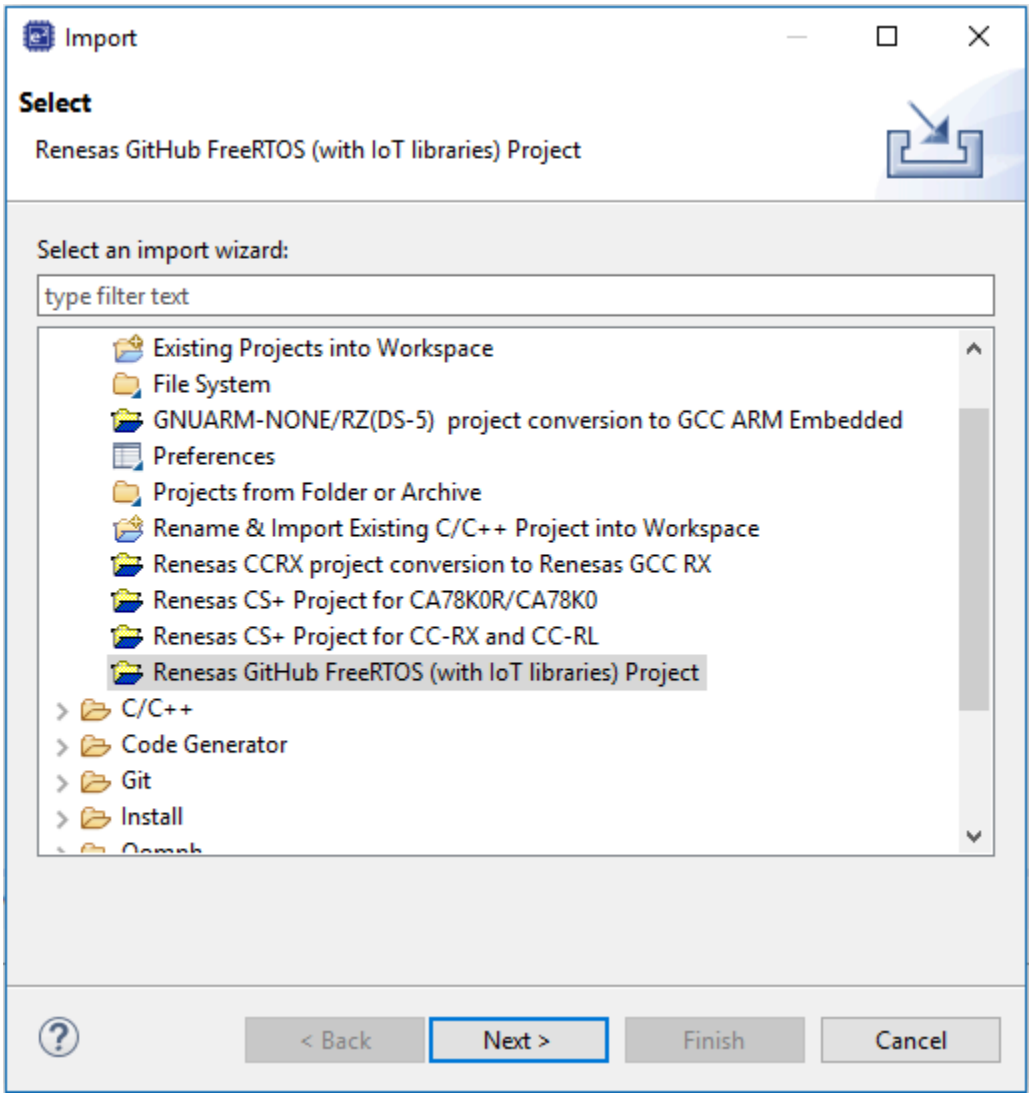
Se não tem o Tera Term instalado no PC, você pode baixá-lo em https://ttssh2.osdn.jp/index.html.en e configurá-lo conforme mostrado aqui. Conecte a porta serial USB do dispositivo ao PC.



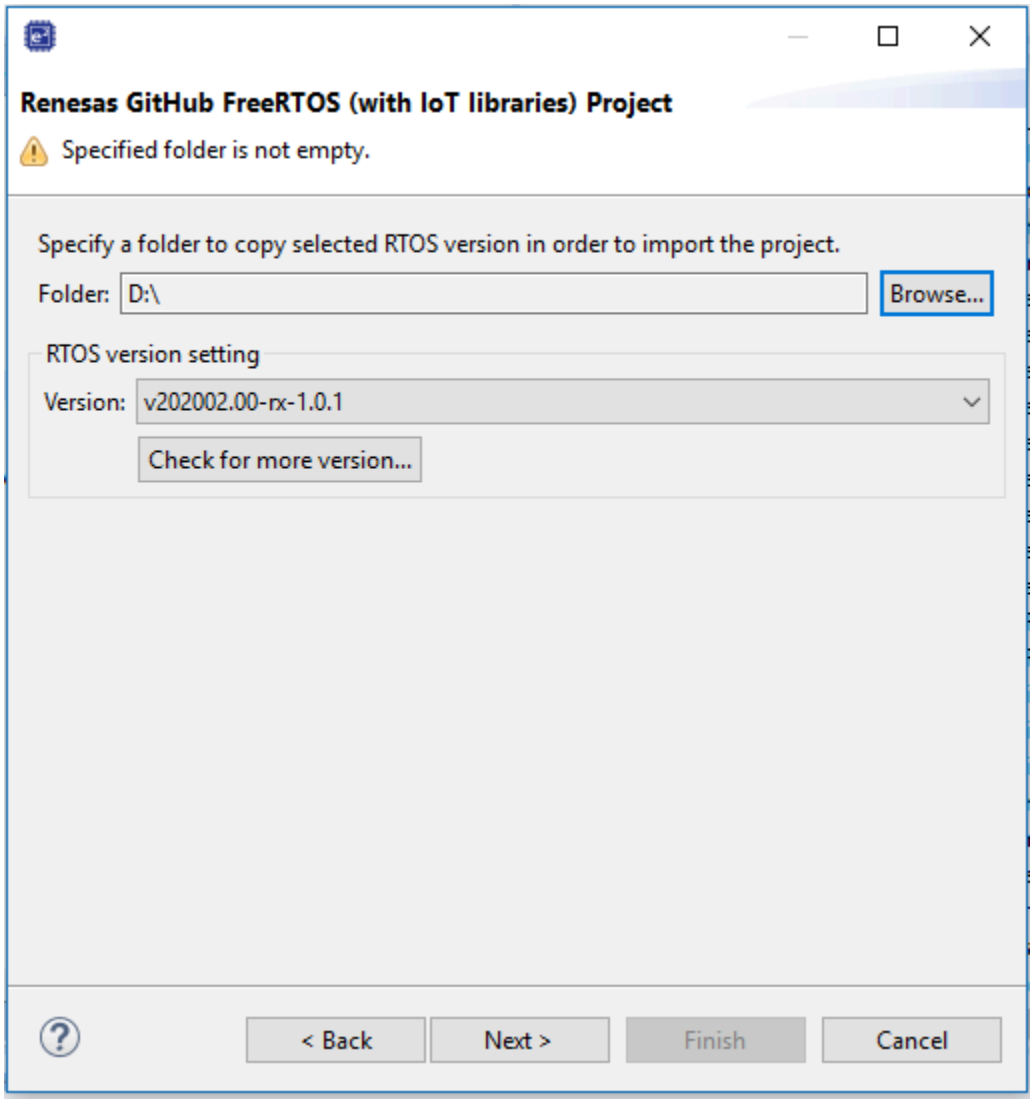
Importe, configure o arquivo de cabeçalho e compile aws_demos e boot_loader

Para começar, você seleciona a versão mais recente do pacote FreeRTOS, e ela será GitHub baixada e importada automaticamente para o projeto. Dessa forma, você pode se concentrar na configuração do FreeRTOS e na escrita do código da aplicação.

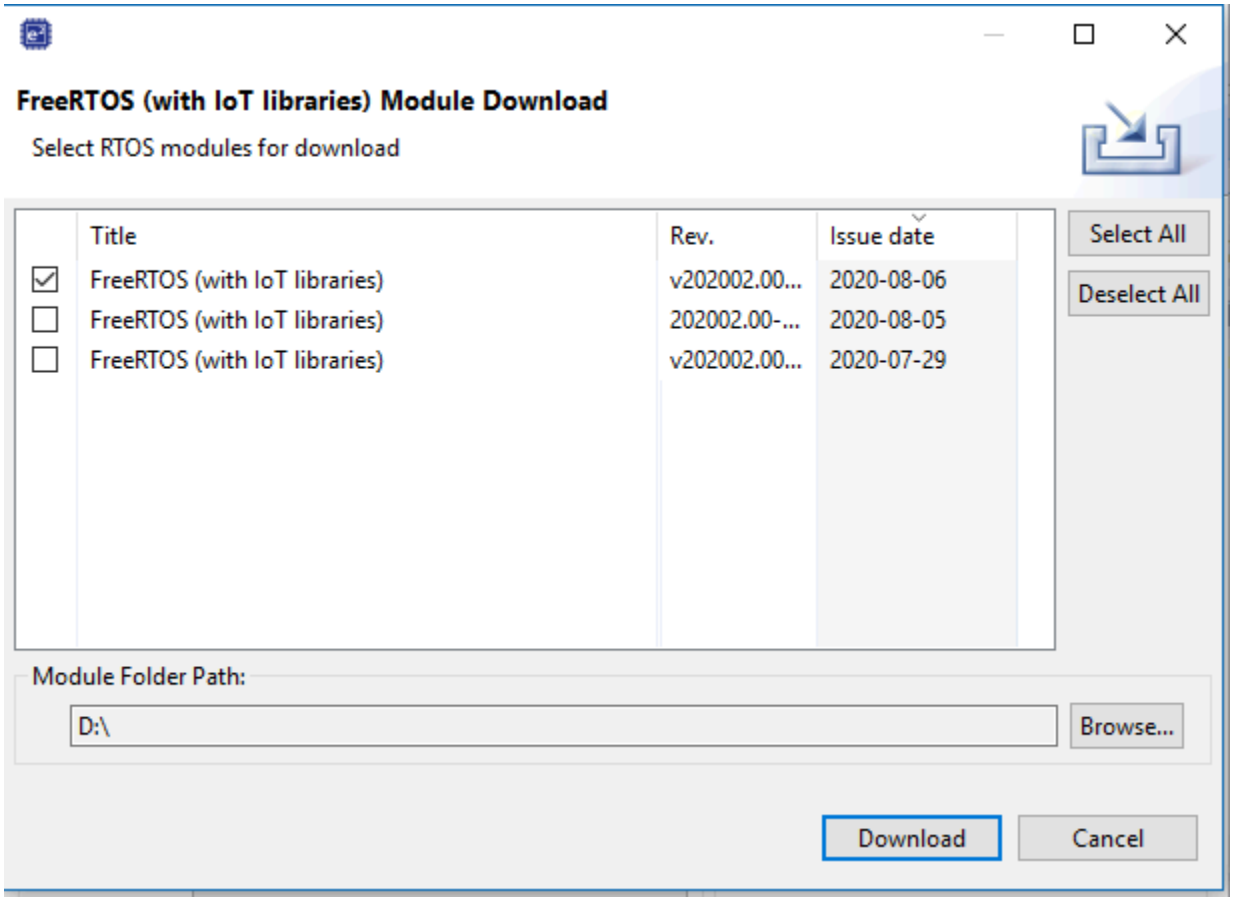
1. Lançamento do e^2 studio.
2. Escolha o Arquivo e depois Importar....
3. Selecione o projeto Renesas GitHub FreeRTOS (com bibliotecas de IoT).



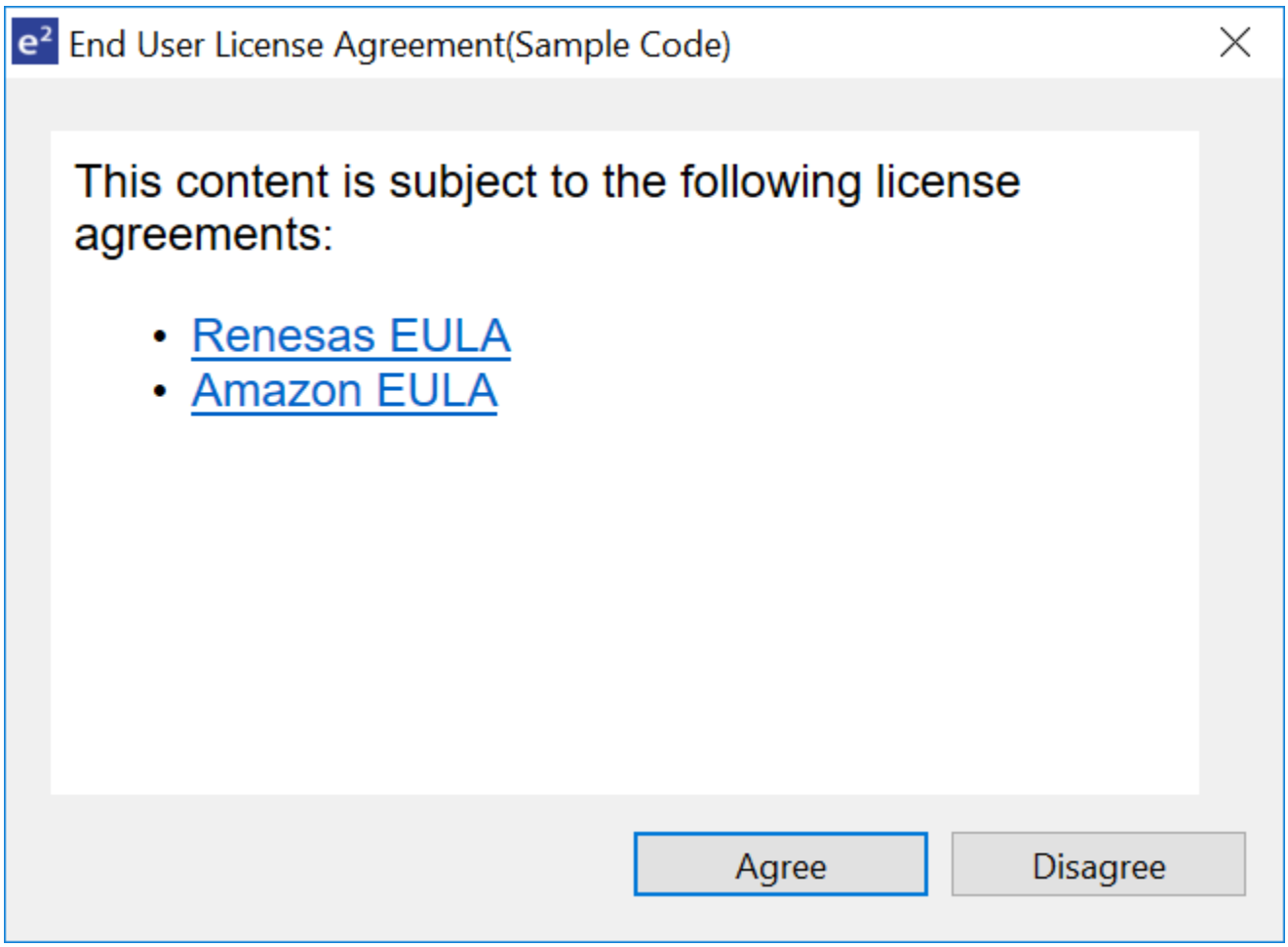
4. Escolha Verificar mais versões... para mostrar a caixa de diálogo de download.



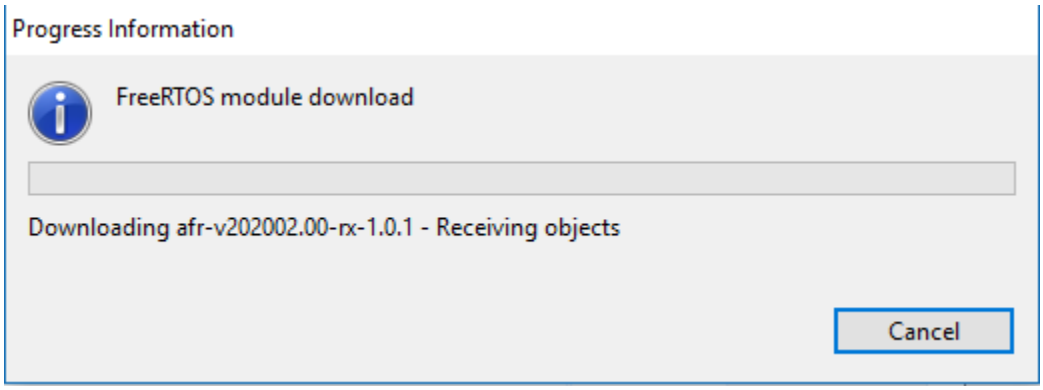
5. Selecione o pacote mais recente.



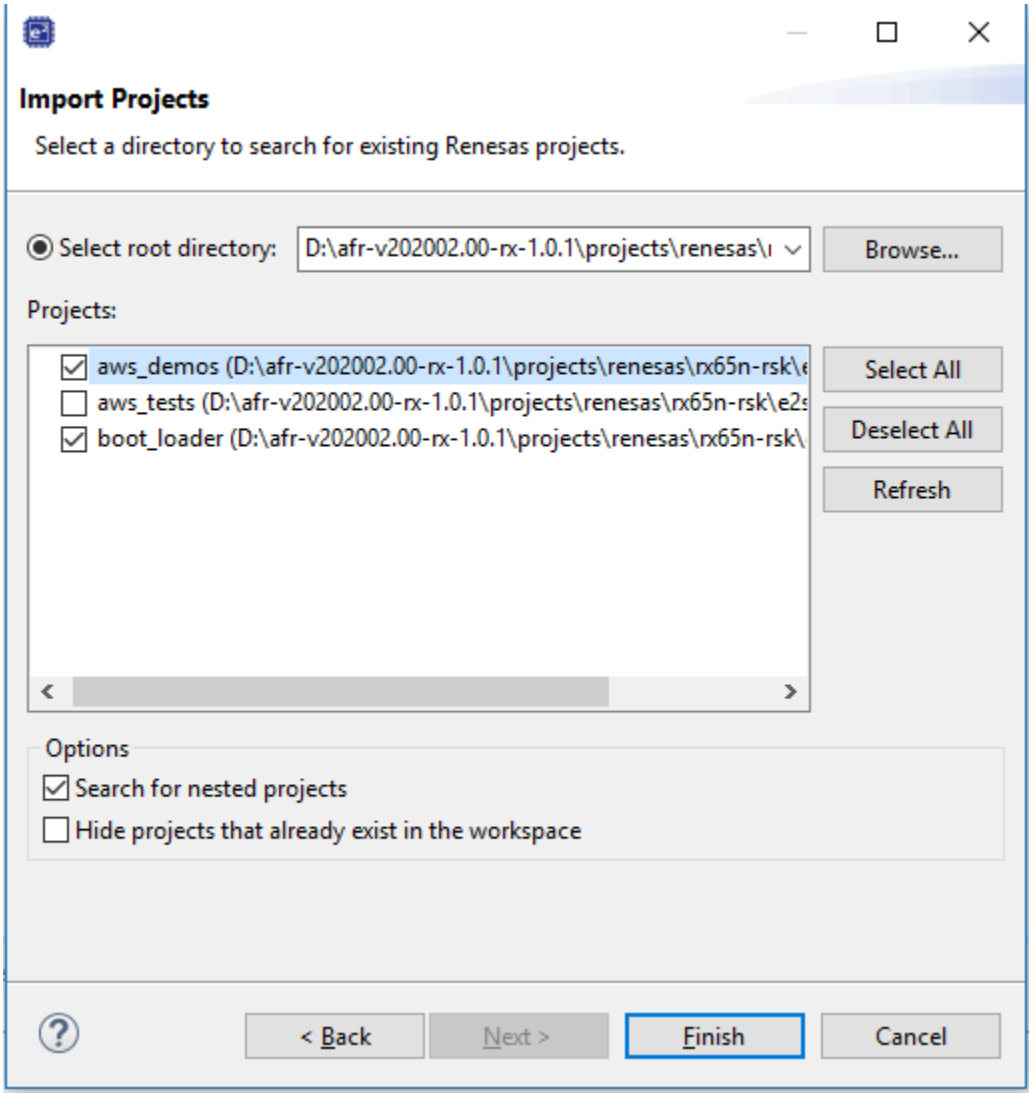
6. Escolha Concordo para aceitar o contrato de licença do usuário final.



7. Aguarde a conclusão do download.



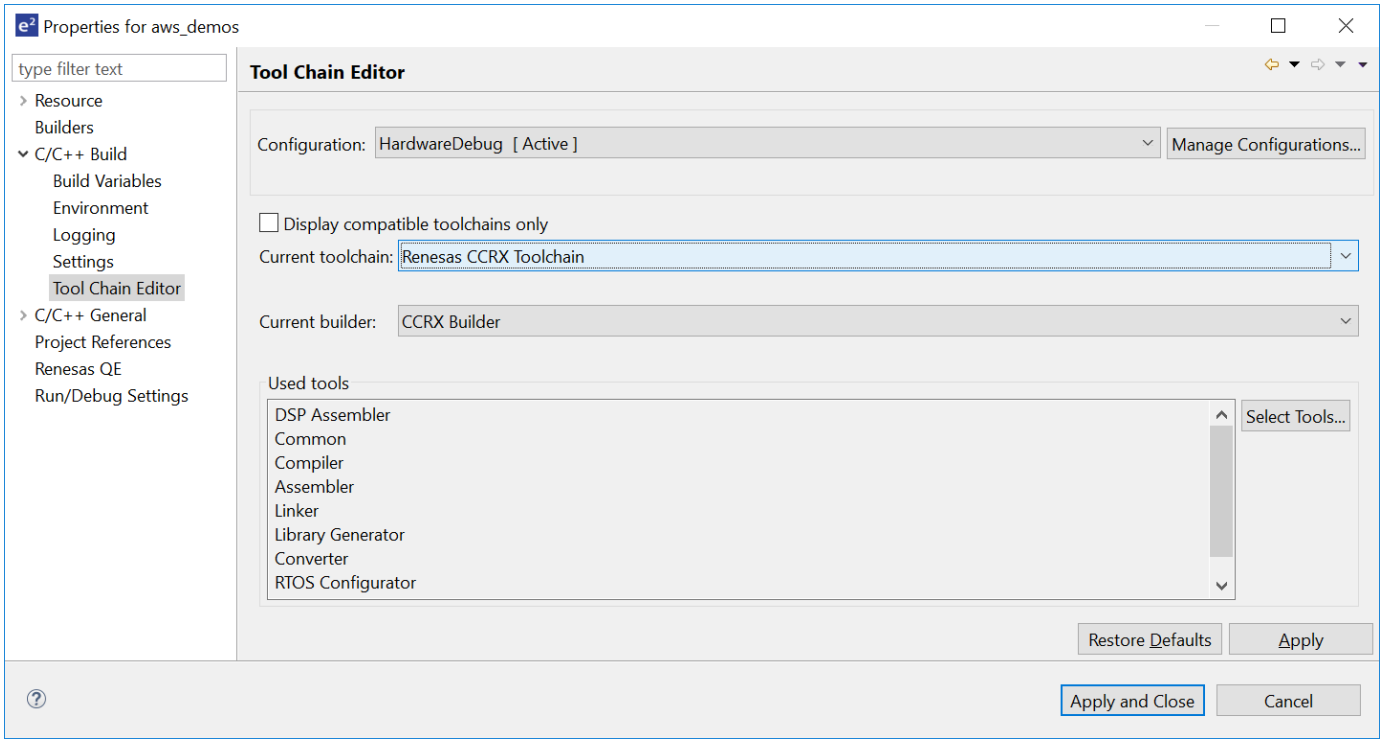
8. Selecione os projetos aws_demos e boot_loader e escolha Concluir para importá-los.



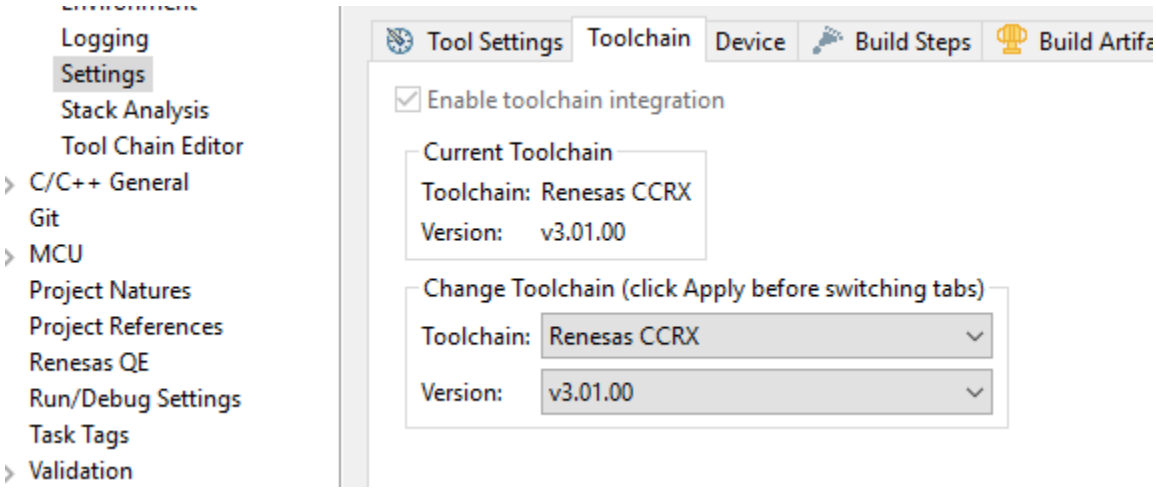
9. Para ambos os projetos, abra as propriedades do projeto. No painel de navegação, selecione Editor de cadeia de ferramentas.

    a. Escolha a Cadeia de ferramentas atual.

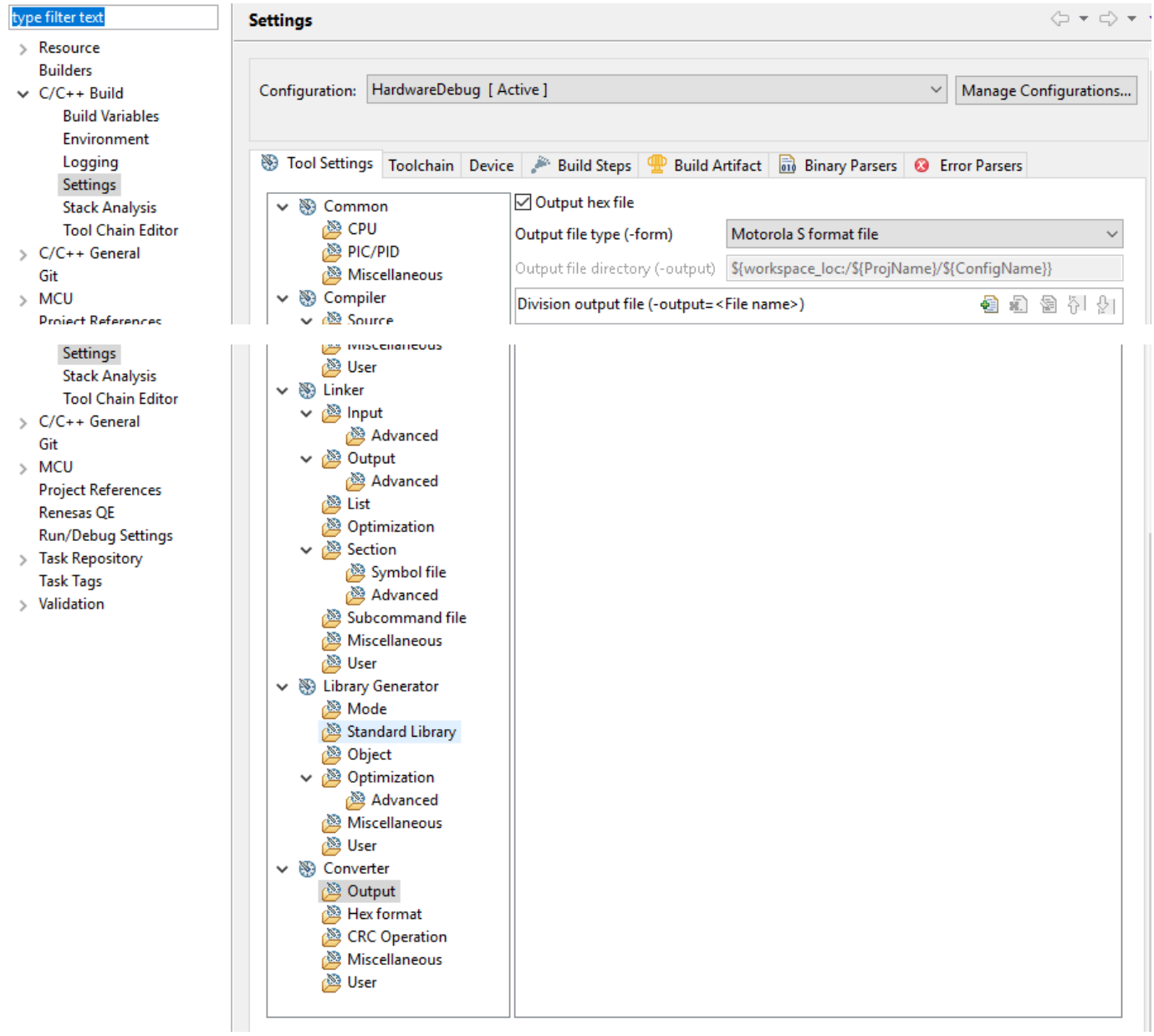
    b. Escolha o Builder atual.



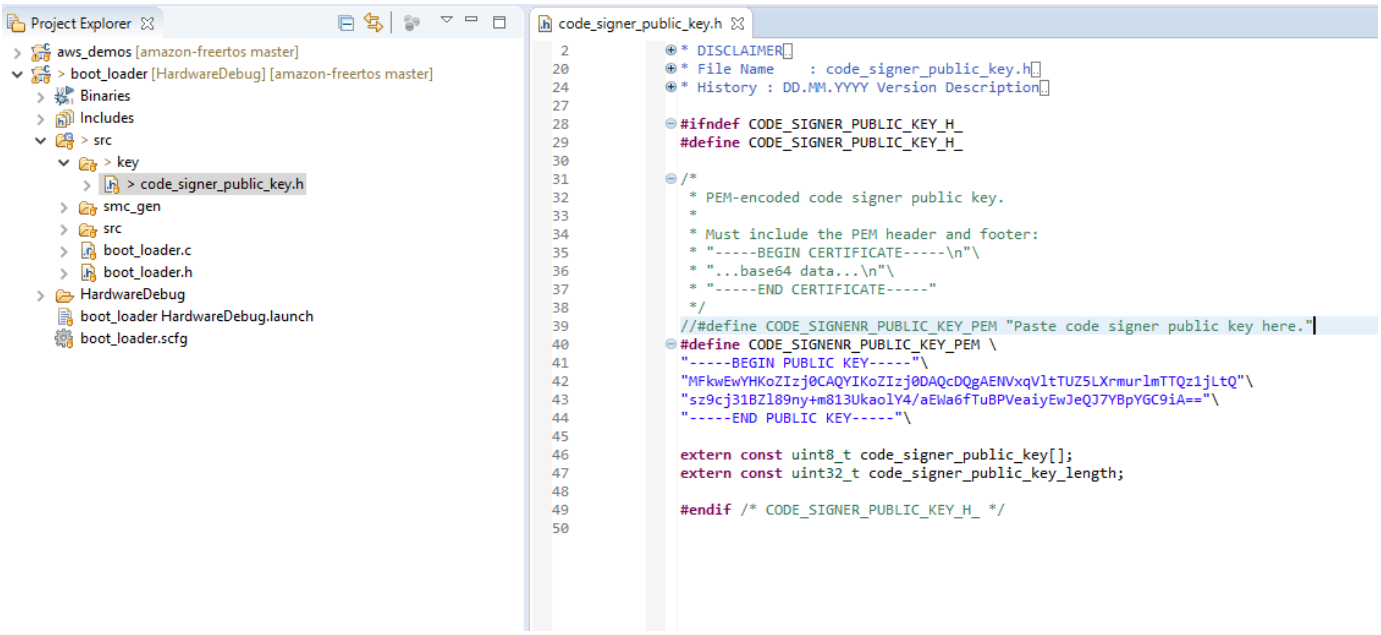
10. No painel de navegação, selecione Configurações. Escolha a guia Cadeia de ferramentas e, em seguida, escolha a Versão da cadeia de ferramentas.



Escolha a guia Configurações de ferramenta, expanda Conversor e escolha Saída. Na janela principal, verifique se Arquivo hexadecimal de saída está selecionado e escolha o tipo de arquivo de saída.



11. No projeto carregador de inicialização, abra `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h` e insira a chave pública. Para obter informações sobre como criar uma chave pública, consulte Como implementar o OTA do FreeRTOS usando o Amazon Web Services no RX65N e a seção 7.3 "Geração de pares de chaves ECDSA-SHA256 com OpenSSL" na Política de criação de atualização de firmware de MCU da Renesas.



Em seguida, compile o projeto para criar `boot_loader.mot`.

12. Abra o projeto `aws_demos`.

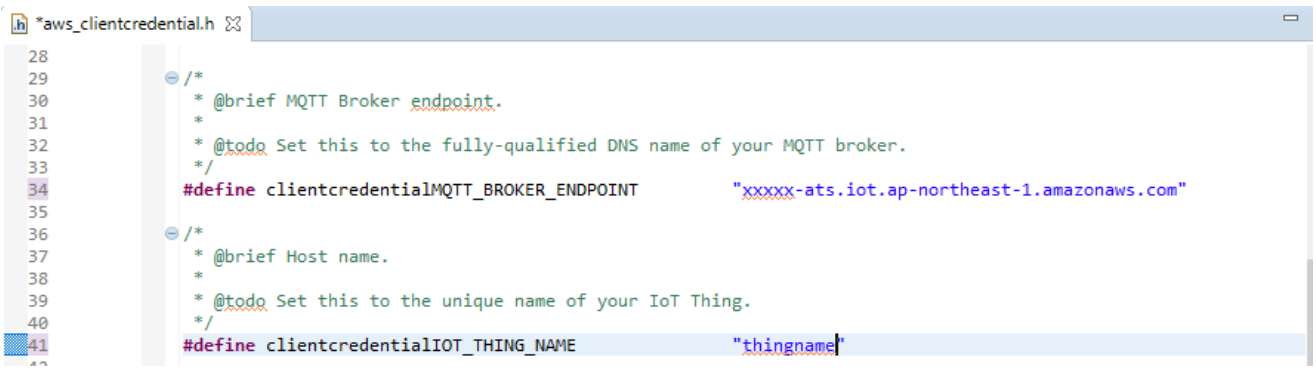
    a. Abra o console de AWS IoT.

    b. No painel de navegação à esquerda, escolha Configurações. Anote seu endpoint personalizado na caixa de texto Endpoint de dados do dispositivo.

    c. Escolha Gerenciar e, em seguida, Coisas. Anote o nome do AWS IoT item do seu quadro.

    d. No projeto `aws_demos`, abra `demos/include/aws_clientcredential.h` e especifique os seguintes valores.

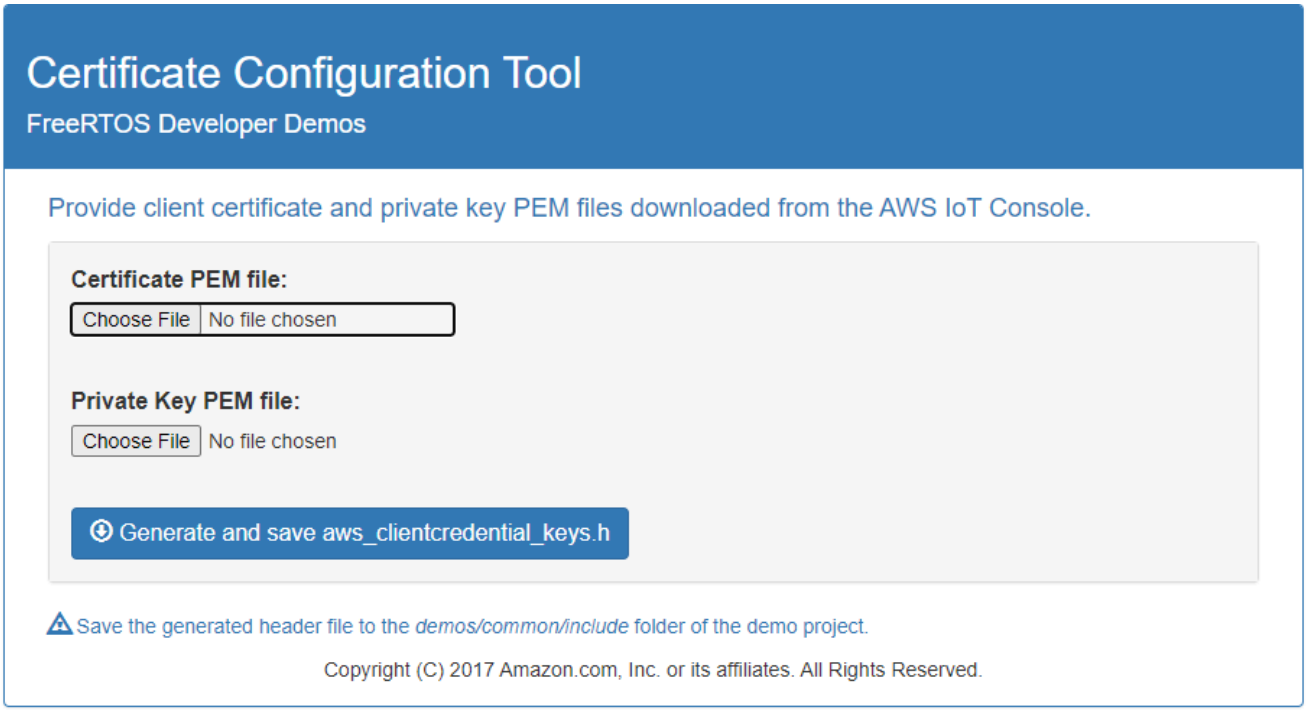
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



e. Abra o arquivo `tools/certificate_configuration/CertificateConfigurator.html`.

f. Importe o arquivo PEM do certificado e o arquivo PEM da chave privada que você baixou anteriormente.

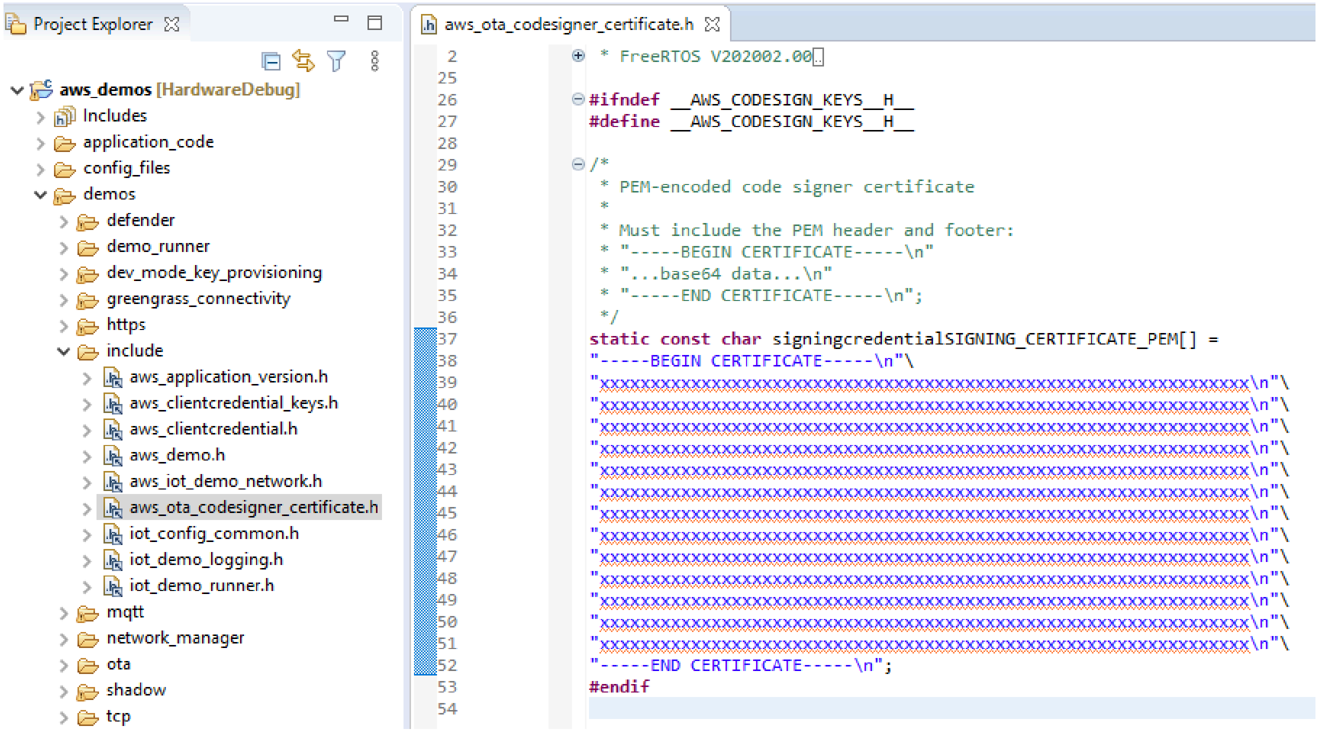
g. Escolha Gerar e salvar aws_clientcredential_keys.h e substitua esse arquivo no diretório `demos/include/`.



h. Abra o arquivo `vendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h` e especifique esses valores.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

No qual *`your-certificate-key`* é o valor do arquivo `secp256r1.crt`. Lembre-se de adicionar "\" após cada linha na certificação. Para obter mais informações sobre a criação do arquivo `secp256r1.crt`, consulte Como implementar o OTA do FreeRTOS usando o Amazon Web Services no RX65N e a seção 7.3 "Geração de pares de chaves ECDSA-SHA256 com OpenSSL" na Política de criação de atualização de firmware de MCU da Renesas.



13. Tarefa A: Instalar a versão inicial do firmware

a. Abra o arquivo `vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` e defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

b. Abra o arquivo `demos/include/ aws_application_version.h` e defina a versão inicial do firmware como `0.9.2`.

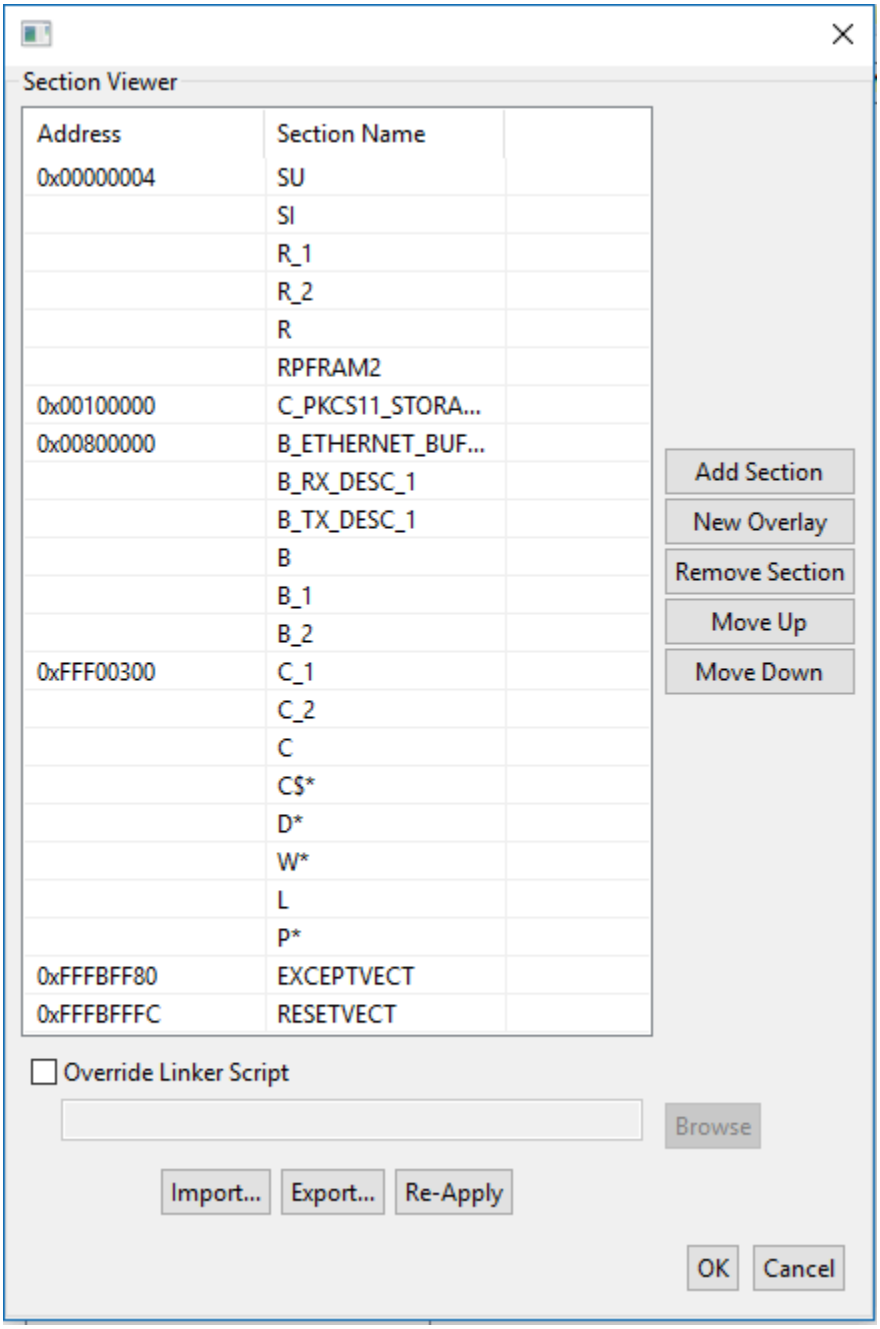
```
#ifndef _AWS_APPLICATION_VERSION_H_
#define _AWS_APPLICATION_VERSION_H_

#include "iot_appversion32.h"
extern const AppVersion32_t xAppFirmwareVersion;

#define APP_VERSION_MAJOR    0
#define APP_VERSION_MINOR    9
#define APP_VERSION_BUILD    2

#endif
```

c. Altere as configurações a seguir em Visualizador de seção.



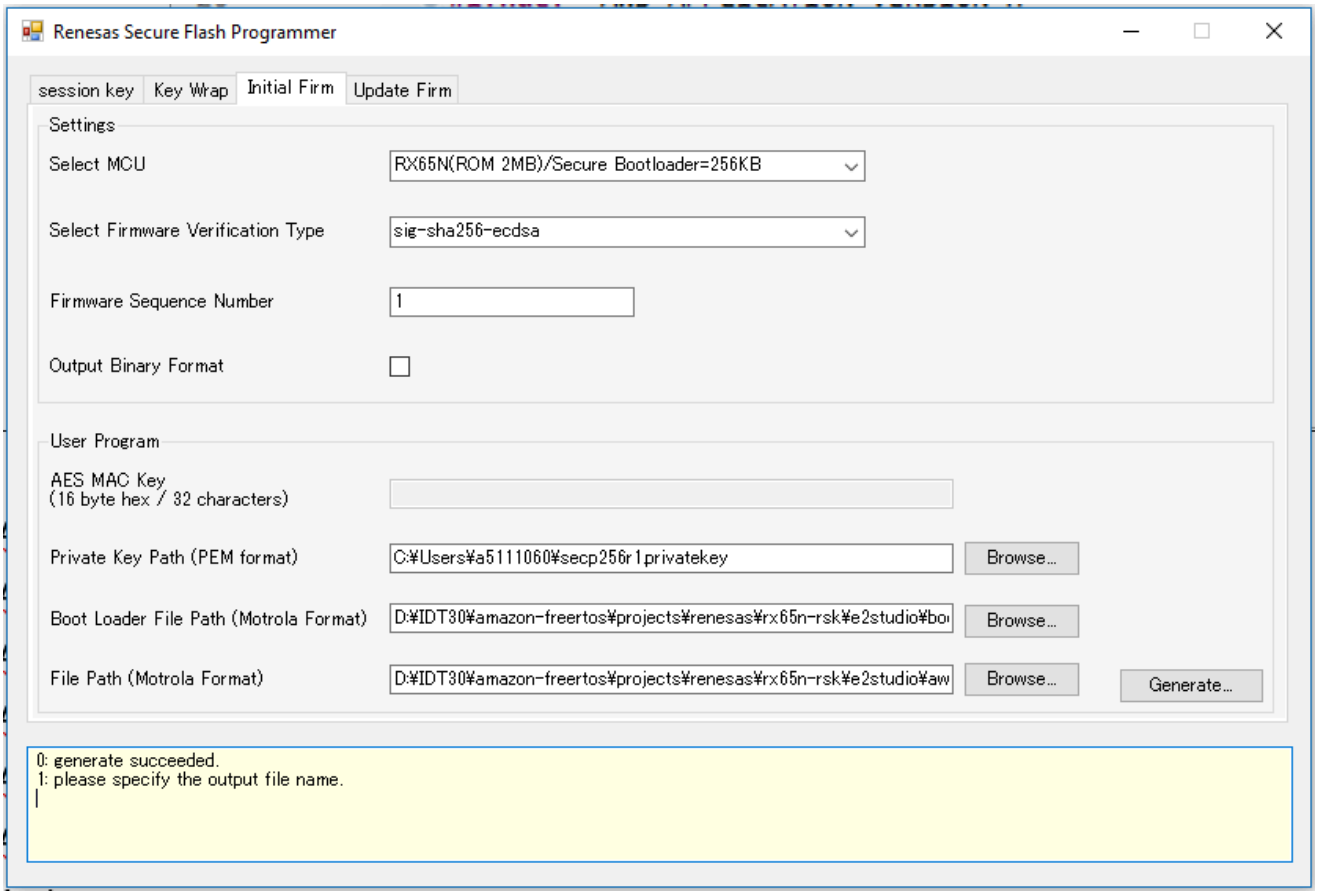
d. Escolha Compilar para criar o arquivo `aws_demos.mot`.

14. Crie o arquivo `userprog.mot` com o Renesas Secure Flash Programmer. `userprog.mot` é uma combinação de `aws_demos.mot` e `boot_loader.mot`. Você pode atualizar esse arquivo para o RX65N-RSK para instalar o firmware inicial.

    a. Baixe https://github.com/renesas/Amazon-FreeRTOS-Tools e abra `Renesas Secure Flash Programmer.exe`.

    b. Escolha a guia Firmware inicial e defina os seguintes parâmetros:

- Caminho da chave privada: a localização de `secp256r1.privatekey`.
- Caminho do arquivo do carregador de inicialização: a localização de `boot_loader.mot` (`projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug`).
- Caminho do arquivo: a localização do arquivo `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).



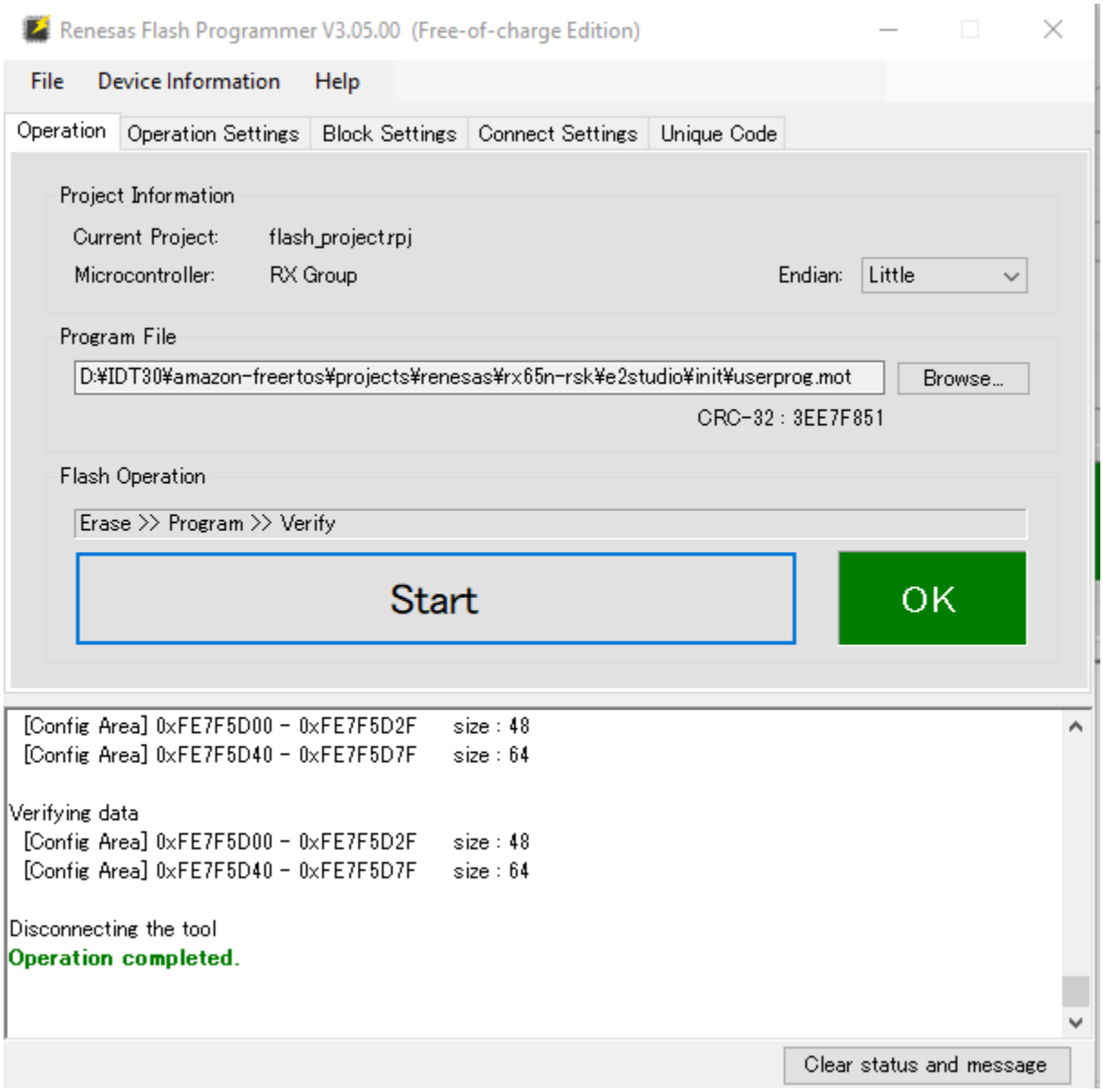
c. Crie um diretório chamado `init_firmware` gere `userprog.mot` e salve-o no diretório `init_firmware`. Verifique se o que foi gerado teve êxito.

15. Atualize o firmware inicial no RX65N-RSK.

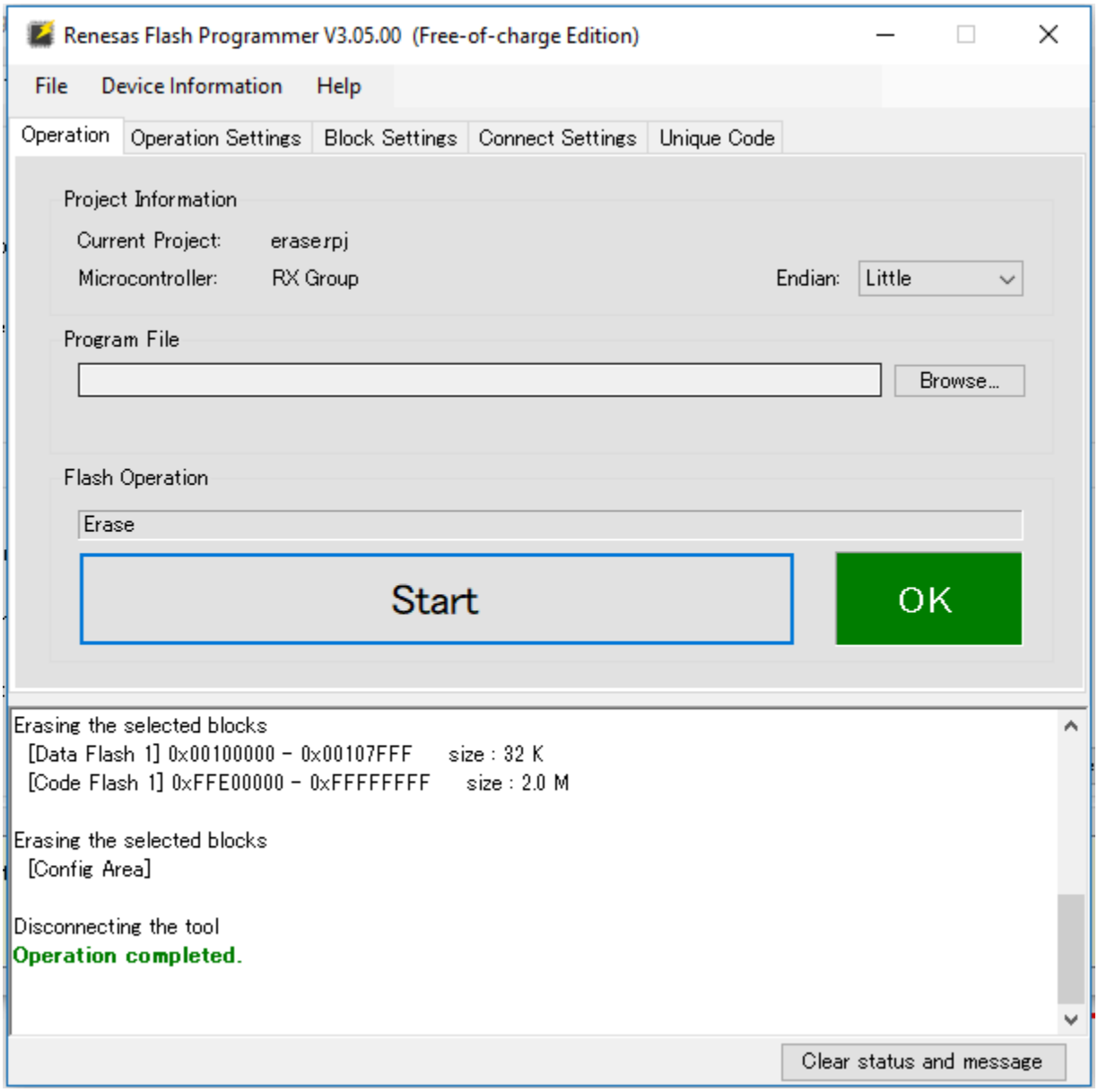
a. Baixe a versão mais recente do Renesas Flash Programmer (GUI de programação) em https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html.

b. Abra o arquivo `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` para apagar dados no banco.

c. Escolha Iniciar para apagar o banco.



d. Para atualizar `userprog.mot`, escolha Procurar... e navegue até o diretório `init_firmware`, selecione o arquivo `userprog.mot` e escolha Iniciar.



16. A versão 0.9.2 (versão inicial) do firmware foi instalada no seu RX65N-RSK. Agora, a placa RX65N-RSK está recebendo as atualizações OTA. Ao abrir o Tera Term no PC, você verá algo parecido com o seguinte quando o firmware inicial for executado.

```
-------------------------------------------------
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. Tarefa B: Atualização da versão do firmware

    a. Abra o arquivo `demos/include/aws_application_version.h` e incremente o valor do token `APP_VERSION_BUILD` para `0.9.3`.

b. Recrie o projeto.

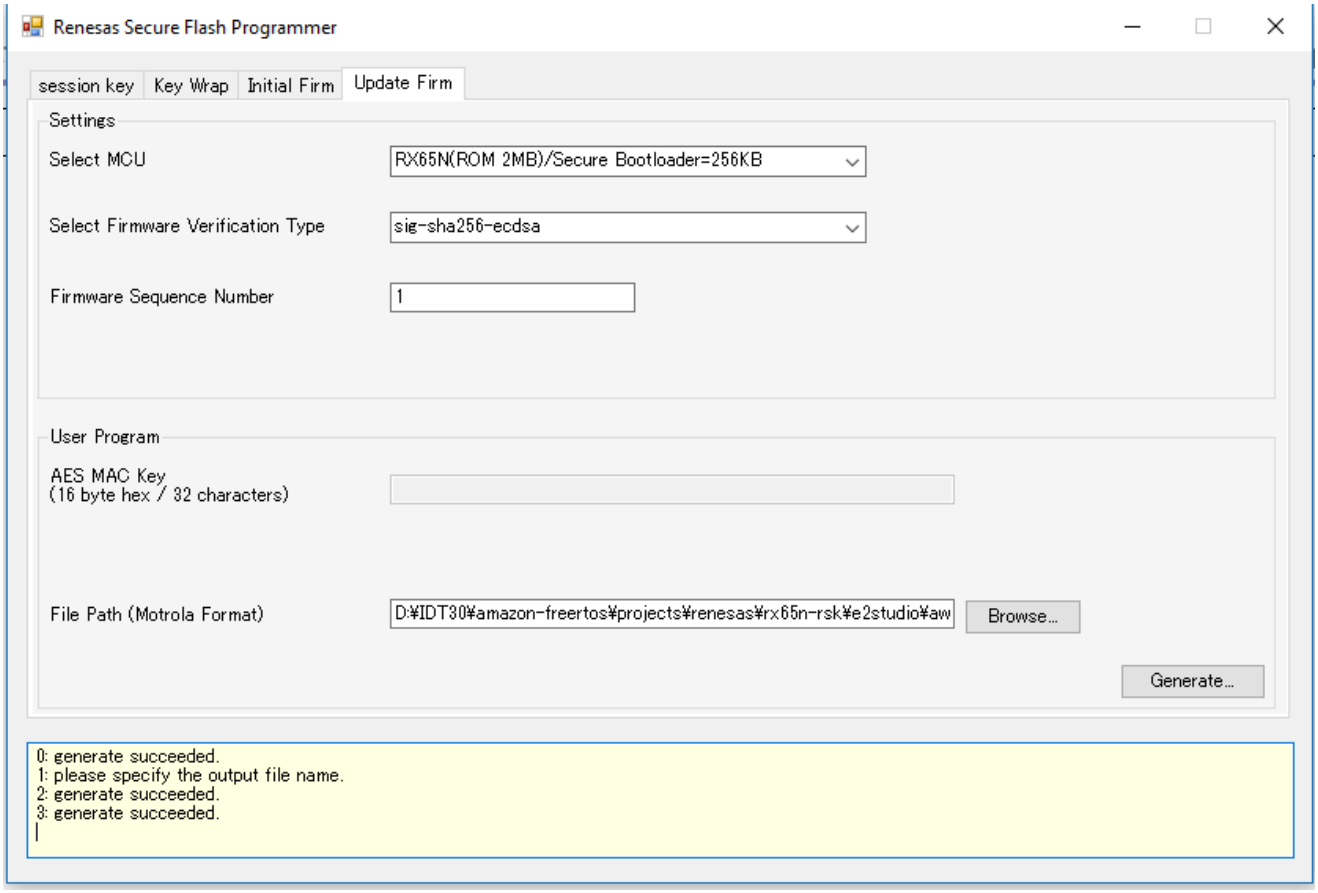
18. Crie o arquivo `userprog.rsu` com o Renesas Secure Flash Programmer para atualizar a versão do seu firmware.

    a. Abra o arquivo `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe`.

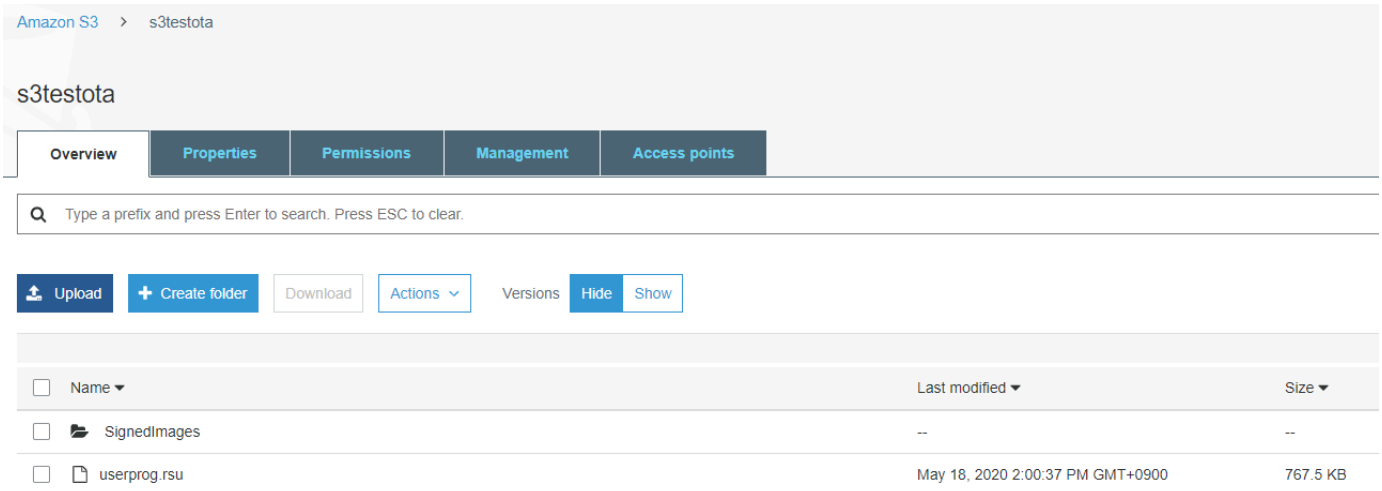
    b. Escolha a guia Atualizar firmware e defina os seguintes parâmetros:

    - Caminho do arquivo: a localização do arquivo `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).

    c. Crie um diretório chamado `update _firmware`. Gere `userprog.rsu` e salve-o no diretório `update_firmware`. Verifique se o que foi gerado teve êxito.



19. Faça o upload da atualização do firmware `userproj.rsu`, em um bucket do Amazon S3, conforme descrito em Criação de um bucket do Amazon S3 para armazenar a atualização.



20. Crie uma trabalho para atualizar o firmware no RX65N-RSK.

    AWS IoT Jobs é um serviço que notifica um ou mais dispositivos conectados sobre um Job pendente. Um trabalho pode ser usado para gerenciar uma frota de dispositivos, atualizar o firmware e os certificados de segurança em dispositivos ou realizar tarefas administrativas, como reiniciar dispositivos e realizar diagnósticos.

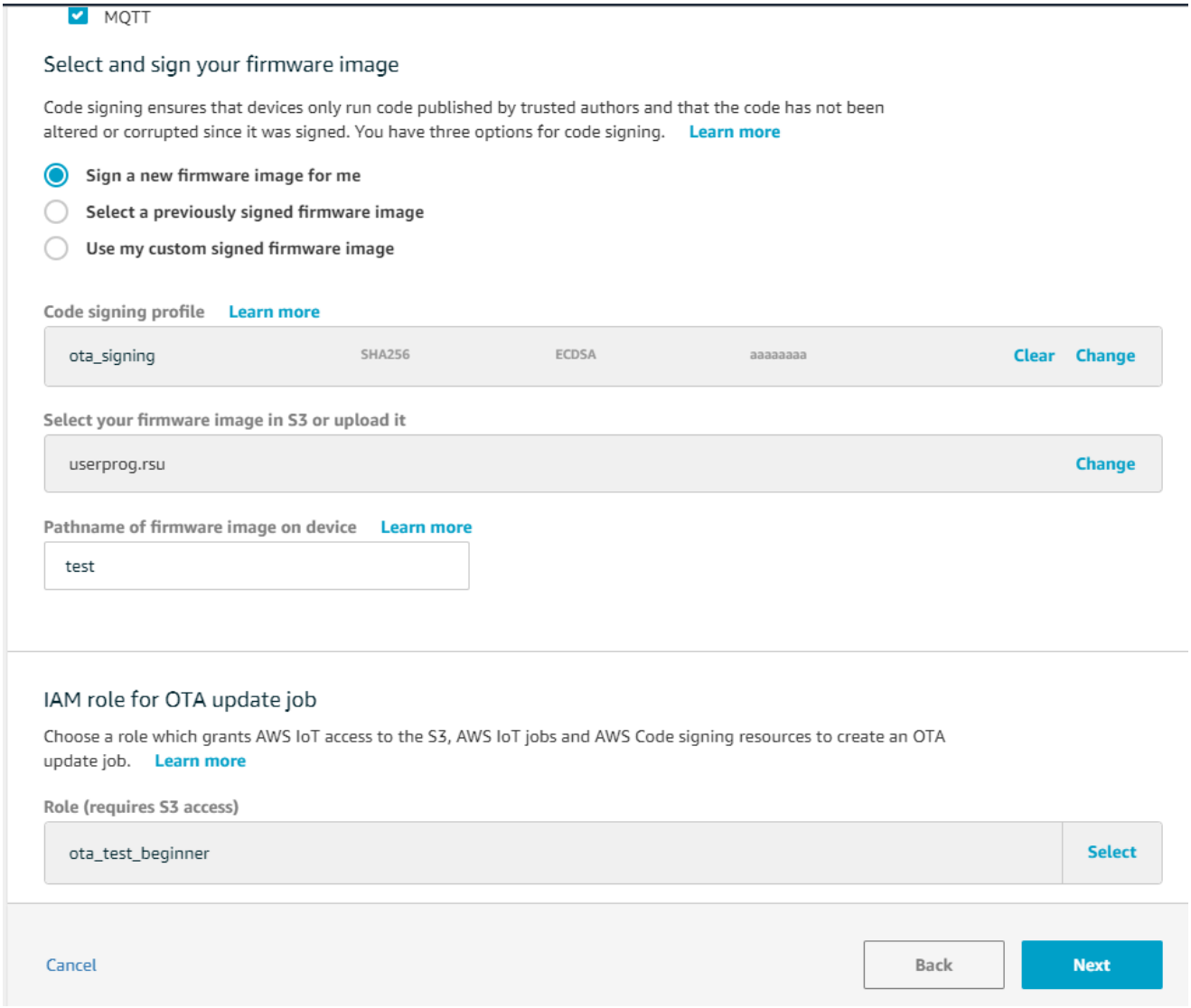
    a. Faça login no console do AWS IoT. No painel de navegação, escolha Gerenciar e escolha Trabalhos.

    b. Escolha Criar um trabalho, e escolha Criar trabalho de atualização OTA. Selecione uma coisa e escolha Avançar.

    c. Crie um trabalho de atualização OTA do FreeRTOS desta forma:

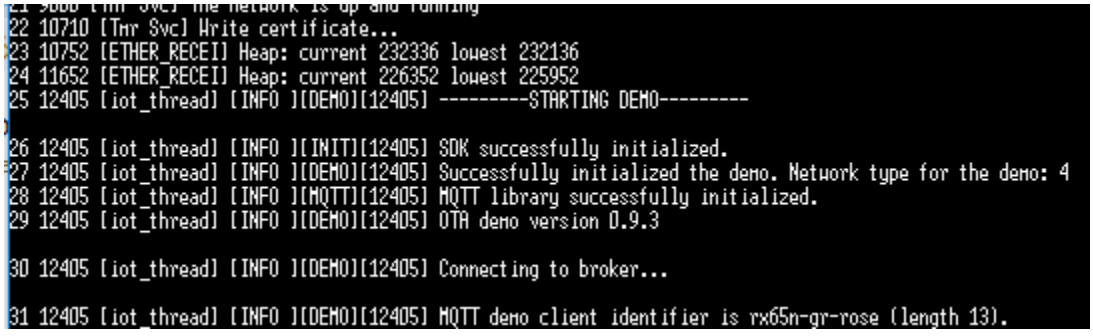
    - Escolha MQTT.
    - Selecione o perfil de assinatura de código criado na seção anterior.
    - Selecione a imagem do firmware que você carregou em um bucket do Amazon S3.
    - Em Nome do caminho da imagem do firmware no dispositivo, insira **`test`**.
    - Selecione o perfil do IAM que você criou na seção anterior.

    d. Escolha Próximo.

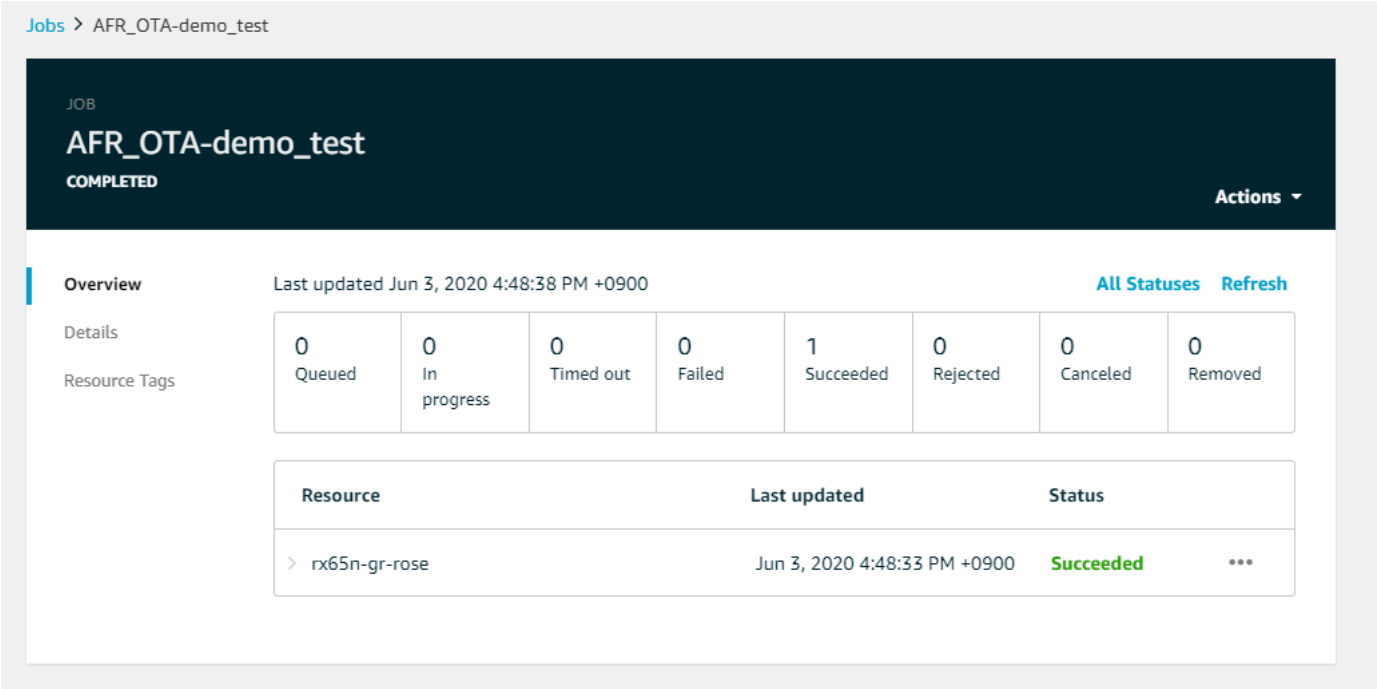


e. Insira um ID e escolha Criar.

21. Reabra o Tera Term para verificar se o firmware foi atualizado com êxito para a versão de demonstração 0.9.3 do OTA.



22. No AWS IoT console, verifique se o status do trabalho é Bem-sucedido.



Tutorial: executar atualizações OTA no Espressif ESP32 usando o Bluetooth Low Energy do FreeRTOS

> **Important**
>
> Essa integração de referência está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Este tutorial mostra como atualizar um microcontrolador Espressif ESP32 conectado a um proxy MQTT do Bluetooth Low Energy em um dispositivo Android. Ele atualiza o dispositivo usando trabalhos de atualização sem fios do AWS IoT. O dispositivo se conecta ao AWS IoT usando as credenciais do Amazon Cognito inseridas no aplicativo de demonstração do Android. Um operador autorizado inicia a atualização OTA a partir da nuvem. Quando o dispositivo se conecta por meio do aplicativo de demonstração do Android, a atualização OTA é iniciada e o firmware é atualizado no dispositivo.

As versões principais e posteriores do FreeRTOS 2019.06.00 incluem suporte a proxy MQTT do Bluetooth Low Energy que pode ser usado para provisionamento de Wi-Fi e conexões seguras aos

serviços do AWS IoT. Ao usar o atributo Bluetooth Low Energy, você pode criar dispositivos de baixo consumo de energia que podem ser emparelhados com um dispositivo móvel para conectividade sem precisar de Wi-Fi. Os dispositivos podem se comunicar usando o MQTT conectando-se por meio de SDKs do Bluetooth Low Energy para Android ou iOS que usam perfis de perfil de acesso genérico (GAP) e atributos genéricos (GATT).

Confira as etapas que seguiremos para permitir atualizações OTA via Bluetooth Low Energy:

1. Configurar o armazenamento: crie um bucket e políticas do Amazon S3 e configure um usuário que possa realizar atualizações.
2. Criar um certificado de assinatura de código: crie um certificado de assinatura e permita que o usuário assine atualizações de firmware.
3. Configurar a autenticação do Amazon Cognito: crie um provedor de credenciais, um grupo de usuários e acesso de aplicativos ao grupo de usuários.
4. Configurar o FreeRTOS: configure o Bluetooth Low Energy, as credenciais do cliente e o certificado público de assinatura de código.
5. Configurar um aplicativo Android: configure o provedor de credenciais, o grupo de usuários e implante o aplicativo em um dispositivo Android.
6. Executar o script de atualização OTA: para iniciar uma atualização OTA, use o script de atualização OTA.

Para obter mais informações sobre como as atualizações funcionam, consulte Atualizações sem fios do FreeRTOS. Para obter informações adicionais sobre como configurar a funcionalidade de proxy MQTT do Bluetooth Low Energy, consulte o post Usar o Bluetooth Low Energy com o FreeRTOS no Espressif ESP32, do Richard Kang.

Pré-requisitos

Para executar as etapas neste tutorial, você precisa dos seguintes recursos:

- Uma placa de desenvolvimento ESP32.
- Um cabo microUSB para USB A.
- Uma conta da AWS (o nível gratuito é suficiente).
- Um telefone Android com Android v 6.0 ou posterior e Bluetooth versão 4.2 ou posterior.

No seu computador de desenvolvimento, você precisa:

- Espaço em disco suficiente (~ 500 Mb) para a cadeia de ferramentas Xtensa e o código-fonte e exemplos do FreeRTOS.
- Android Studio instalado.
- A AWS CLI instalada.
- Python3 instalado.
- O kit de desenvolvimento de software (SDK) da AWS boto3 para Python.

As etapas deste tutorial consideram que a cadeia de ferramentas Xtensa, o ESP-IDF e o código FreeRTOS estão instalados no diretório inicial `/esp`. Você deve adicionar `~/esp/xtensa-esp32-elf/bin` à sua variável `$PATH`.

Etapa 1: configurar o armazenamento

1. Criação de um bucket do Amazon S3 para armazenar a atualização com o versionamento ativado para armazenar as imagens do firmware.
2. Criação de uma função de serviço de atualização de OTA e adicione as políticas gerenciadas a seguir ao perfil:
   - AWSIotLogging
   - AWSIoTRuleActions
   - AWSIoTThingsRegistration
   - AWSFreeRTOSOTAUpdate
3. Criar um usuário que possa realizar atualizações OTA. Esse usuário pode assinar e implantar atualizações de firmware em dispositivos IoT na conta e tem acesso para fazer atualizações OTA em todos os dispositivos. O acesso deve ser limitado a entidades confiáveis.
4. Siga as etapas em Criação de uma política de usuário de OTA e anexe-o ao seu usuário.

Etapa 2: criar o certificado de assinatura de código

1. Crie um bucket do Amazon S3 com o versionamento habilitado para armazenar as imagens de firmware.
2. Crie um certificado de assinatura de código que possa ser usado para assinar o firmware. Observe o nome do recurso da Amazon (ARN) do certificado quando o certificado for importado.

```
aws acm import-certificate --profile=ota-update-user --certificate file://
ecdsasigner.crt --private-key file://ecdsasigner.key
```

Exemplos de resultado:

```
{
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"
}
```

Você usará o ARN mais tarde para criar um perfil de assinatura. Se desejar, você pode criar o perfil agora com o seguinte 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
```

Exemplos de resultado:

```
{
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"
}
```

Etapa 3: configuração da autenticação do Amazon Cognito

Criar uma política do AWS IoT

1. Faça login no console do AWS IoT.
2. No canto superior direito do console, escolha Minha conta. Em Configurações da conta, anote o ID de conta de 12 dígitos.
3. No painel de navegação à esquerda, escolha Settings (Configurações). Em Endpoint de dados do dispositivo, anote o valor do endpoint. O endpoint deve ser algo com `xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com`. Neste exemplo, a região da AWS é definida como "us-west-2".

4. No painel de navegação, escolha Segurança, Políticas e Criar. Se você não tiver nenhuma política em sua conta, verá a mensagem "Você ainda não tem nenhuma política" e poderá escolher Criar uma política.
5. Insira um nome para a sua política, por exemplo, "esp32_mqtt_proxy_iot_policy".
6. Na seção Add statements (Adicionar instruções), escolha Advanced mode (Modo avançado). Copie e cole o seguinte JSON na janela do editor de política. Substitua `aws-account-id` pelo ID da sua conta e `aws-region` pela sua região (por exemplo, "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. Escolha Create (Criar).

Crie uma coisa do AWS IoT

1. Faça login no console do AWS IoT.
2. No painel de navegação à esquerda, escolha Manage (Gerenciar) e, depois, Things (Coisas).

3. No canto superior direito, escolha Criar. Se você não tiver coisas registradas em sua conta, a mensagem "Você ainda não tem coisas registradas" é exibida e você poderá escolher Registrar uma coisa.
4. Na página Creating AWS IoT things (Criar coisas para IoT), escolha Create a single thing (Criar uma única coisa).
5. Na página Adicionar o dispositivo ao registro de coisa, insira um nome para a coisa (por exemplo, "esp32-ble"). São permitidos somente caracteres alfanuméricos, hífen (-) e sublinhados (_). Escolha Next (Próximo).
6. Na página Adicionar um certificado para a coisa, em Ignorar certificado e criar coisa, escolha Criar uma coisa sem certificado. Como usamos o aplicativo móvel proxy BLE que usa uma credencial do Amazon Cognito para autenticação e autorização, certificado de dispositivo é necessário.

Criação de um cliente de aplicação do Amazon Cognito

1. Faça login no console do Amazon Cognito.
2. No banner de navegação superior direito, escolha Criar um grupo de usuários.
3. Insira o nome do gurpo (por exemplo, "esp32_mqtt_proxy_user_pool").
4. Escolha Review defaults.
5. Em Clientes de aplicação, escolha Adicionar cliente de aplicação e, em seguida, escolha Adicionar um cliente de aplicação.
6. Insira um nome de cliente de aplicação (por exemplo, "mqtt_app_client").
7. Verifique se a opção Gerar segredo do cliente está selecionada.
8. Escolha Create app client (Criar cliente da aplicação).
9. Escolha Return to pool details (Retornar aos detalhes do grupo).
10. Na página Revisar do grupo de usuários, selecione Criar grupo. Você receberá uma mensagem dizendo: "O grupo de usuários foi criado com êxito". Anote o ID do grupo.
11. No painel de navegação à esquerda, escolha Clientes de aplicação.
12. Escolha Mostrar detalhes. Anote o ID e o segredo do cliente de aplicação.

Criar um grupo de identidades do Amazon Cognito

1. Faça login no console do Amazon Cognito.

2. Escolha Create new identity pool (Criar novo grupo de identidades).
3. Insira um nome para o banco de identidades (por exemplo, "mqtt_proxy_identity_pool").
4. Expanda os Provedores de autenticação.
5. Escolha a guia Cognito.
6. Insira o ID do grupo de usuários e do cliente de aplicação que você anotou nas etapas anteriores.
7. Escolha Create Pool (Criar grupo).
8. Na próxima página, para criar os perfis para identidades autenticadas e não autenticadas , escolha Permitir.
9. Anote o ID do banco de identidades, que está no formato `us-east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Anexar uma política do IAM à identidade autenticada

1. Abra o console do Amazon Cognito.
2. Selecione o banco de identidades que você acabou de criar (por exemplo, "mqtt_proxy_identity_pool").
3. Escolha Edit identity pool (Editar grupo de identidades).
4. Anote o perfil do IAM atribuío ao perfil autenticada (por exemplo, "Cognito_MQTT_Proxy_Identity_PoolAuth_Role").
5. Abra o console do IAM.
6. No painel de navegação, escolha Roles.
7. Pesquise o perfil atribuído (por exemplo, "Cognito_MQTT_Proxy_Identity_PoolAuth_Role") e selecione-o.
8. Escolha Adicionar política em linha e, em seguida, JSON.
9. Insira a seguinte política:

```
{
       "Version": "2012-10-17",
       "Statement": [
       {
          "Effect": "Allow",
          "Action": [
             "iot:AttachPolicy",
```

```
            "iot:AttachPrincipalPolicy",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe"
        ],
        "Resource": "*"
    }]
}
```

10. Escolha Revisar política.
11. Insira um nome de política (por exemplo, "mqttProxyCognitoPolicy").
12. Escolha Create policy (Criar política).

Etapa 4: configurar o Amazon FreeRTOS

1. Baixe a versão mais recente do código do Amazon FreeRTOS no repositório do FreeRTOS no GitHub.
2. Para habilitar a demonstração da atualização OTA, siga as etapas em Introdução ao Espressif ESP32- DevKit C e ao ESP-WROVER-KIT.
3. Faça essas modificações adicionais nos seguintes arquivos:

   a. Abra `vendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h` e defina `CONFIG_OTA_UPDATE_DEMO_ENABLED`.

   b. Abra `vendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h` e mude `democonfigNETWORK_TYPES` para `AWSIOT_NETWORK_TYPE_BLE`.

   c. Abra `demos/include/aws_clientcredential.h` e insira o URL do endpoint para `clientcredentialMQTT_BROKER_ENDPOINT`.

      Insira o nome do item para `clientcredentialIOT_THING_NAME` (por exemplo, "esp32-ble"). Os certificados não precisam ser adicionados quando você usa as credenciais do Amazon Cognito.

   d. Abra `vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h`, altere `configSUPPORTED_NETWORKS` e `configENABLED_NETWORKS` para inclur somente `AWSIOT_NETWORK_TYPE_BLE`.

   e. Abra o arquivo `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` e insira seu certificado.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

A aplicação deve iniciar e imprimir a versão de demonstração:

```
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...
```

Etapa 5: configurar um aplicativo para Android

1. Baixe o SDK do Bluetooth Low Energy para Android e um aplicativo de amostra do repositório amazon-freertos-ble-android-sdk do GitHub.
2. Abra o arquivo `app/src/main/res/raw/awsconfiguration.json` e preencha o ID do grupo, a região, o appClientID e o appClientSecret usando as instruções no exemplo de JSON a seguir.

```
{
  "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 insira o nome da política que você criou anteriormente (por exemplo, *`esp32_mqtt_proxy_iot_policy`*) e também a Região (por exemplo, *`us-east-1`*).
4. Compilar e instalar o aplicativo de demonstração.
   a. No Android Studio, escolha Compilar e, em seguida, Aplicativo Make Module.
   b. Selecione Executar > Executar aplicativo. Você pode acessar o painel da janela do Logcat no Android Studio para monitorar as mensagens de log.
   c. No dispositivo Android, crie uma conta na tela de login.
   d. Criar um usuário. Se já existe um usuário, insira as credenciais.
   e. Permita que a demonstração do Amazon FreeRTOS acesse a localização do dispositivo.
   f. Verificar dispositivos Bluetooth Low Energy.
   g. Mova o controle deslizante do dispositivo encontrado para Ativado.
   h. Pressione S no console de depuração da porta serial do ESP32.
   i. Escolha Emparelhar e conectar.
5. O link Mais... ficará ativo depois que a conexão for estabelecida. O estado da conexão deverá mudar para "BLE_CONNECTED" no logcat do dispositivo Android quando a conexão for concluída:

```
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 que as mensagens possam ser transmitidas, o dispositivo Amazon FreeRTOS e o dispositivo Android negociam a MTU. A seguinte saída deverá ser mostrada no logcat:

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD:
 onMTUChanged : 512 status: Success
```

7. O dispositivo se conecta ao aplicativo e começa a enviar mensagens MQTT usando o proxy MQTT. Para confirmar se o dispositivo pode se comunicar, verifique se o valor dos dados característicos do MQTT_CONTROL mudam para 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. Quando os dispositivos estiverem emparelhados, você receberá um aviso no console ESP32. Para ativar o BLE, pressione S. A demonstração não funcionará até que você realize essa etapa.

```
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.
```

Etapa 6: executar o script de atualização OTA

1. Para instalar os pré-requisitos, execute os seguintes comandos:

```
pip3 install boto3
```

```
pip3 install pathlib
```

2. Incremente a versão do aplicativo FreeRTOS em `demos/include/aws_application_version.h`.
3. Compile um arquivo.bin novo.
4. Baixe o script python start_ota.py. Para visualizar o conteúdo da ajuda do script, execute o seguinte comando em uma janela de terminal:

```
python3 start_ota.py -h
```

Você deve ver algo parecido com o exemplo a seguir:

```
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. Se você usou o modelo AWS CloudFormation fornecido para criar recursos, execute o seguinte 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
```

Você deve visualizar o início da atualização no console de depuração do 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. Quando a atualização do OTA for concluída, o dispositivo será reiniciado conforme exigido pelo processo de atualização OTA. Em seguida, ele tenta se conectar usando o firmware atualizado. Se a atualização for bem-sucedida, o firmware atualizado será marcado como ativo e você deverá ver a versão atualizada no console:

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

Aplicativo de demonstração do Device Shadow da AWS IoT

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

Introdução

Esta demonstração mostra como usar a biblioteca do AWS IoT Device Shadow para se conectar ao serviço do AWS Device Shadow. Ela usa o Biblioteca coreMQTT para estabelecer uma conexão MQTT com TLS (autenticação mútua) com o agente MQTT do AWS IoT e o analisador da biblioteca coreJSON para analisar documentos paralelos recebidos do serviço de Sombra da AWS. A demonstração mostra operações básicas de sombra, como atualizar um documento de sombra e como excluir um documento de sombra. A demonstração também mostra como registrar uma função de retorno de chamada na biblioteca coreMQTT para lidar com mensagens como a `/update` de sombra e as mensagens de `/update/delta` enviadas do serviço do AWS IoT Device Shadow.

Esta demonstração serve apenas como um exercício de aprendizado porque a solicitação para atualizar o documento de sombra (estado) e a resposta de atualização são feitas pelo mesmo aplicativo. Em um cenário de produção realista, um aplicativo externo solicitaria uma atualização do estado do dispositivo remotamente, mesmo que o dispositivo não estivesse conectado no momento. O dispositivo reconhecerá a solicitação de atualização quando estiver conectado.

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.

Funcionalidade

A demonstração cria uma tarefa de aplicativo única que faz loop por um conjunto de exemplos que demonstram `/update` de sombras e `/update/delta` de retornos de chamada para simular a alternância do estado de um dispositivo remoto. Ela envia uma atualização de sombra com o novo estado `desired` e espera o dispositivo mudar o estado `reported` em resposta ao novo estado `desired`. Além disso, um retorno de chamada de `/update` de sombra é usado para exibir as mudanças dos estados de sombra. Essa demonstração também usa uma conexão MQTT segura com o agente MQTT do AWS IoT e considera que há um estado `powerOn` na sombra do dispositivo.

A demonstração realiza as seguintes operações:

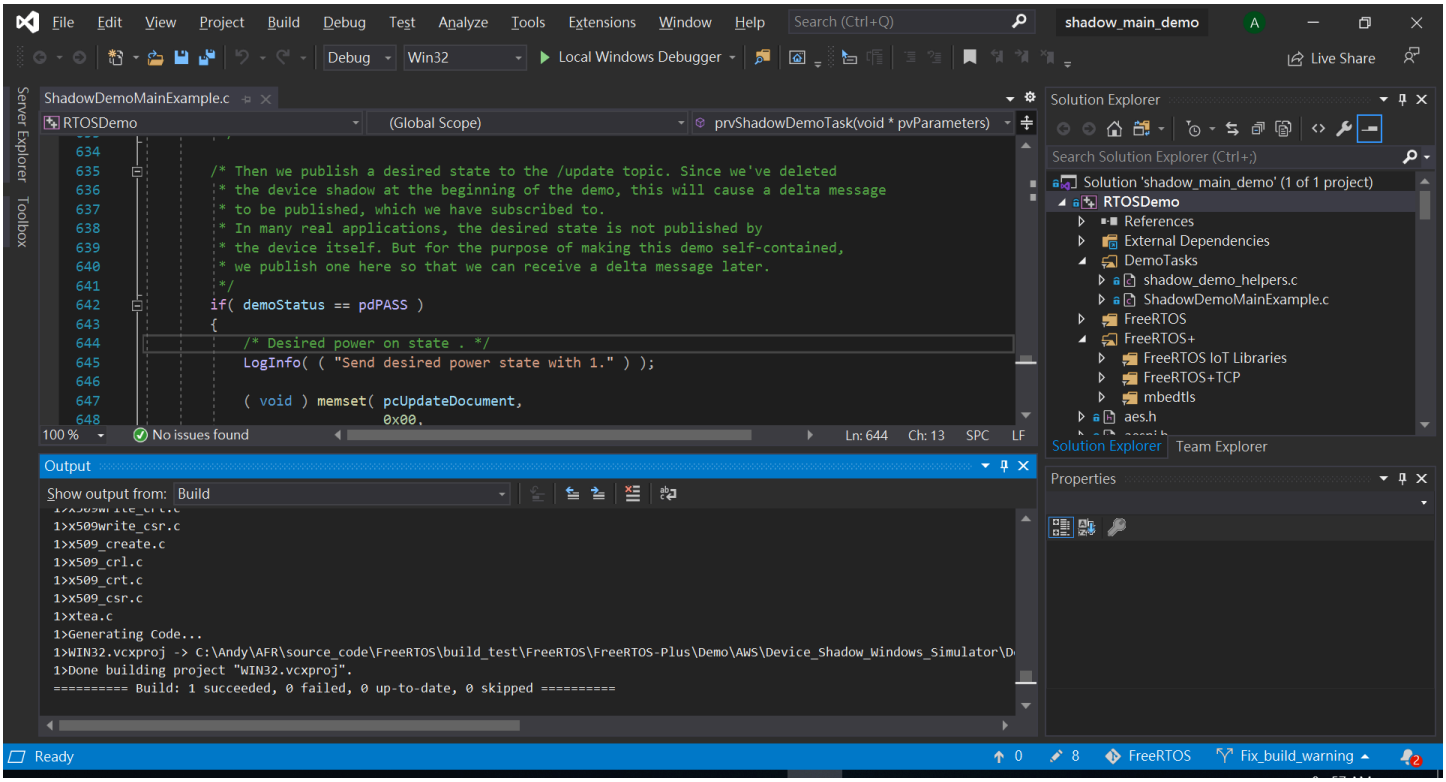
1. Estabelece uma conexão MQTT usando as funções auxiliares em `shadow_demo_helpers.c`.
2. Monta strings de tópicos MQTT para operações de sombra do dispositivo, usando macros definidas pela biblioteca do AWS IoT Device Shadow.

3. Publica no tópico MQTT usado para excluir uma sombra do dispositivo para excluir toda sombra do dispositivo existente.
4. Assina os tópicos MQTT para `/update/delta`, `/update/accepted` e `/update/rejected` usando funções auxiliares em `shadow_demo_helpers.c`.
5. Publica um estado desejado de `powerOn` usando funções auxiliares em `shadow_demo_helpers.c`. Isso fará com que uma mensagem de `/update/delta` seja enviada ao dispositivo.
6. Gerencie as mensagens MQTT recebidas em `prvEventCallback` e determine se a mensagem está relacionada à sombra do dispositivo usando uma função definida pela biblioteca do AWS IoT Device Shadow (`Shadow_MatchTopic`). Se a mensagem for uma mensagem de `/update/delta` da sombra do dispositivo, a função de demonstração principal publicará uma segunda mensagem para atualizar o estado relatado como `powerOn`. Se uma mensagem de `/update/accepted` for recebida, verifique se ela tem o mesmo `clientToken` publicado anteriormente na mensagem de atualização. Isso marcará o fim da demonstração.

```
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]
```

A demonstração pode ser encontrada no arquivo *freertos*`/demos/device_shadow_for_aws/shadow_demo_main.c` ou no GitHub.

A captura de tela a seguir mostra a saída esperada quando a demonstração é bem-sucedida.



Conecte-se ao agente MQTT da AWS IoT

Para nos conectarmos ao agente MQTT do AWS IoT, usamos o mesmo método que `MQTT_Connect()` na Demonstração de autenticação mútua da coreMQTT.

Exclusão de documento de sombra

Para excluir o documento de sombra, chame `xPublishToTopic` com uma mensagem vazia, usando macros definidas pela biblioteca do AWS IoT Device Shadow. Isso usa `MQTT_Publish` para publicar no tópico `/delete`. A seção de código a seguir mostra como isso é feito na função `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 );
```

Assinatura em tópicos de sombra

Assine os tópicos do Device Shadow para receber notificações do agente do AWS IoT sobre mudanças nas sombras. Os tópicos do Device Shadow são montados por macros definidas na biblioteca do Device Shadow. A seção de código a seguir mostra como isso é feito na função `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 ) );
}
```

Envio de atualizações de sombras

Para enviar uma atualização de sombra, a demonstração chama `xPublishToTopic` com uma mensagem no formato JSON, usando macros definidas pela biblioteca do Device Shadow. Isso usa `MQTT_Publish` para publicar no tópico `/delete`. A seção de código a seguir mostra como isso é feito na função `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 ) );
```

Lidar com mensagens delta de sombra e mensagens de atualização de sombra

A função de retorno de chamada do usuário, que foi registrada na biblioteca de clientes coreMQTT usando a função `MQTT_Init`, notificará sobre um evento de pacote de entrada. Veja a função de retorno de chamada `prvEventCallback` no GitHub.

A função de retorno de chamada confirma que o pacote de entrada é do tipo `MQTT_PACKET_TYPE_PUBLISH` e usa a API `Shadow_MatchTopic` da biblioteca do Device Shadow para confirmar se a mensagem recebida é uma mensagem de sombra.

Se a mensagem recebida for uma mensagem de sombra com tipo `ShadowMessageTypeUpdateDelta`, chamaremos `prvUpdateDeltaHandler` para lidar com essa mensagem. O manipulador `prvUpdateDeltaHandler` usa a biblioteca coreJSON para analisar a mensagem e obter o valor delta do estado `powerOn` e o compara com o estado atual do dispositivo mantido localmente. Se forem diferentes, o estado do dispositivo local será atualizado para refletir o novo valor do estado `powerOn` do documento de sombra.

Se a mensagem recebida for uma mensagem de sombra com tipo `ShadowMessageTypeUpdateAccepted`, chamaremos `prvUpdateAcceptedHandler` para lidar com essa mensagem. O manipulador `prvUpdateAcceptedHandler` analisa a mensagem usando a biblioteca coreJSON para obter o `clientToken` da mensagem. Essa função de manipulador verifica se o token do cliente da mensagem JSON corresponde ao token do cliente usado pelo aplicativo. Se não corresponder, a função registrará uma mensagem de aviso.

Demonstração do cliente Echo de Secure Sockets

> **Important**
>
> Essa demonstração está hospedada no repositório Amazon-FreeRTOS, que está preterido. Recomendamos começar aqui ao criar um novo projeto. Se você já tem um projeto FreeRTOS existente baseado no repositório Amazon-FreeRTOS que está preterido, consulte o Guia de migração do repositório Github do Amazon FreeRTOS.

O exemplo a seguir usa uma única tarefa do RTOS. O código-fonte para este exemplo pode ser encontrado em `demos/tcp/aws_tcp_echo_client_single_task.c`.

Antes de começar, verifique se você fez download do FreeRTOS no seu microcontrolador, criou e executou os projetos de demonstração do FreeRTOS. Você pode clonar ou fazer download do FreeRTOS do GitHub. Consulte o arquivo README.md para obter instruções.

Para executar a demonstração

> **Note**
>
> Para configurar e executar as demonstrações do FreeRTOS, siga as etapas em Conceitos básicos do FreeRTOS.
> No momento, os Kits de desenvolvimento Cypress CYW943907AEVAL1F e CYW954907AEVAL1F não oferecem suporte às demonstrações de cliente e servidor de TCP.

1. Siga as instruções em Configuração do TLS Echo Server no Guia de portabilidade do FreeRTOS.

   Um servidor echo TLS deve estar em execução e na escuta na porta 9000.

   Durante a configuração, você deverá gerar quatro arquivos:

   - `client.pem` (certificado de cliente)
   - `client.key` (chave privada de cliente)
   - `server.pem` (certificado de servidor)
   - `server.key` (chave privada de servidor)

2. Use a ferramenta `tools/certificate_configuration/CertificateConfigurator.html` para copiar o certificado de cliente (`client.pem`) e a chave privada de cliente (`client.key`) para `aws_clientcredential_keys.h`.
3. Abra o arquivo `FreeRTOSConfig.h`.
4. Defina as variáveis `configECHO_SERVER_ADDR0`, `configECHO_SERVER_ADDR1`, `configECHO_SERVER_ADDR2` e `configECHO_SERVER_ADDR3` como os quatro inteiros que compõem o endereço IP em que o servidor Echo TLS está em execução.
5. Defina a variável `configTCP_ECHO_CLIENT_PORT` como `9000`, a porta em que o servidor Echo TLS está escutando.
6. Defina a variável `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` como `1`.
7. Use a ferramenta `tools/certificate_configuration/PEMfileToCString.html` para copiar o certificado de servidor (`server.pem`) para `cTlsECHO_SERVER_CERTIFICATE_PEM` no arquivo `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` e defina `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

O microcontrolador e o servidor Echo TLS devem estar na mesma rede. Quando a demonstração começar (`main.c`), você verá a mensagem de log `Received correct string from echo server`.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.