aws

Guia do usuário

FreeRTOS

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

FreeRTOS: Guia do usuário

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

As marcas comerciais e imagens comerciais da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é o FreeRTOS? 1
Download do código-fonte do FreeRTOS 1
Versionamento do FreeRTOS 2
Arquitetura do FreeRTOS 2
Plataformas de hardware qualificadas para o FreeRTOS 3
Fluxo de trabalho de desenvolvimento 4
Recursos adicionais 4
Princípios básicos do kernel do FreeRTOS 5
Programador de kernel do FreeRTOS 6
Gerenciamento de memória 6
Alocação de memória do kernel 6
Gerenciamento de memória de aplicações 7
Coordenação entre tarefas 7
Filas 7
Semáforos e mutexes 8
Direct-to-task notificações 8
Buffers de fluxo 9
Buffers de mensagens 11
Temporizadores de software 12
Suporte para baixo consumo 13
`FreeRTOSConfig.h` 13
AWS IoT SDK de dispositivo para C incorporado 15
Conceitos básicos do FreeRTOS 17
Aplicação de demonstração do FreeRTOS 17
Primeiras etapas 18
Configurando sua AWS conta e permissões 19
Registrando sua placa MCU com AWS IoT 21
Fazer download do FreeRTOS 24
Configuração das demonstrações do FreeRTOS 25
Solução de problemas 27
Dicas gerais da solução de problemas de conceitos básicos 28
Instalação de um emulador de terminal 28
Usando CMake com FreeRTOS 29
Pré-requisitos 30

Desenvolvimento de aplicações do FreeRTOS com ferramentas de terceiros para edição e depuração de código ... 32
Criando FreeRTOS com CMake ... 32
Provisionamento de chaves no modo de desenvolvedor ... 40
Introdução ... 40
Opção #1: importação de chave privada de AWS IoT ... 40
Opção 2: geração de chaves privadas integradas ... 40
Guias de conceitos básicos específicos da placa ... 43
Kit de desenvolvimento Cypress CYW9439 07 AEVAL1 F ... 44
Kit de desenvolvimento Cypress CYW9549 07 AEVAL1 F ... 49
Cypress CY8 CKIT-064S0S2-4343W ... 55
Microchip ATECC6 08A Secure Element com simulador de Windows ... 61
Espressif ESP32 - DevKit C e o ESP-WROVER-KIT ... 69
Expressif - ESP32 WROOM-32SE ... 113
Kit de conectividade IoT Infineon XMC48 00 ... 129
Kit de conectividade XMC48 IoT Infineon OPTIGA Trust X e 00 ... 136
Kit inicial Marvell MW32x AWS IoT ... 144
MediaTek MT7697Kit de desenvolvimento Hx ... 166
Microchip Curiosity PIC32 MZ EF ... 171
Nórdico em RF5284 0-DK ... 178
Nuvoton 487 NuMaker-IoT-M ... 184
Módulo LPC54 IoT NXP 018 ... 192
Renesas Starter Kit+ para N-2MB RX65 ... 197
STMicroelectronics STM32Node IoT do Kit L4 Discovery ... 203
Texas Instruments CC322 0SF-LAUNCHXL ... 206
Simulador de dispositivo do Windows ... 212
Kit de IoT industrial Xilinx Avent MicroZed ... 217
Over-the-Air Atualizações ... 227
Marcação de recursos de OTA ... 228
Pré-requisitos de atualização do OTA ... 228
Criação de um bucket do Amazon S3 para armazenar a atualização ... 229
Criação de uma função de serviço de atualização de OTA ... 230
Criação de uma política de usuário de OTA ... 232
Criação de um certificado de assinatura de código ... 233
Conceda acesso à assinatura de código para AWS IoT ... 245
Faça download do FreeRTOS com a biblioteca OTA ... 246

Pré-requisitos para atualizações de OTA usando MQTT 246
Pré-requisitos para atualizações de OTA usando HTTP 250
Tutorial do OTA 255
Instalação do firmware inicial 256
Atualização da versão do firmware 270
Criando uma atualização OTA (AWS IoT console) 271
Criando uma atualização OTA com o AWS CLI 276
Serviço do gerenciador de atualização OTA 303
Integração do agente OTA à aplicação 304
Gerenciamento de conexões 305
Demonstração OTA simples 305
Como usar um retorno de chamada personalizado para eventos de conclusão de OTA 308
Segurança de OTA 310
Assinatura de código para AWS IoT 311
Solução de problemas do OTA 311
Configurar CloudWatch registros para atualizações do OTA 312
Registre chamadas da API AWS IoT OTA com AWS CloudTrail 317
Obtenha Criar detalhes da OTAUpdate falha usando o AWS CLI 321
Obtenha códigos de falha OTA com o AWS CLI 323
Solução de problemas de atualizações de OTA de vários dispositivos 324
Solucione problemas de atualizações OTA com o Texas Instruments CC322 0SF Launchpad 325
Bibliotecas do FreeRTOS 326
Bibliotecas de portabilidade do FreeRTOS 326
Bibliotecas de aplicações do FreeRTOS 328
Bibliotecas comuns do FreeRTOS 330
Configuração das bibliotecas do FreeRTOS 331
Bibliotecas comuns 331
Operações atômicas 332
Contêineres lineares 333
Registro em log 333
Memória estática 334
Grupo de tarefas 334
backoffAlgorithm 339
Introdução 339
Bluetooth Low Energy 340

Visão geral 340
Arquitetura 341
Dependências e requisitos 343
Arquivo de configuração da biblioteca 344
Otimização 344
Restrições de uso 345
Inicialização 346
Referência da API 348
Exemplo de uso 348
Portabilidade 352
Celular SDKs para dispositivos Bluetooth FreeRTOS 355
E/S comum 356
AWS IoT Device Defender 357
Introdução 358
AWS IoT Greengrass 359
Visão geral 359
Dependências e requisitos 359
Referência da API 360
Exemplo de uso 360
coreHTTP 362
Introdução 362
coreJSON 363
Introdução 363
Uso de memória 363
coreMQTT 364
Introdução 364
Agente OTA 366
Visão geral 366
Atributos 366
Referência da API 367
Exemplo de uso 367
Portabilidade 368
Uso de memória 368
núcleo PKCS11 369
Visão geral 369
Atributos 370

Suporte ao sistema criptográfico assimétrico 371
Portabilidade 373
Uso de memória 373
Secure Sockets 374
Visão geral 374
Dependências e requisitos 375
Atributos 375
Solução de problemas 376
Suporte ao desenvolvedor 376
Restrições de uso 376
Inicialização 377
Referência da API 377
Exemplo de uso 377
Portabilidade 379
AWS IoT Device Shadow 380
Introdução 380
AWS IoT Empregos 381
Introdução 381
Transport Layer Security 382
Wi-Fi 382
Visão geral 382
Dependências e requisitos 383
Atributos 383
Configuração 385
Inicialização 385
Referência da API 386
Exemplo de uso 386
Portabilidade 389
Demonstrações do FreeRTOS 390
Execução da demonstração do FreeRTOS 390
Configuração das demonstrações 391
Bluetooth Low Energy 391
Visão geral 391
Pré-requisitos 392
Componentes comuns 395
MQTT por Bluetooth Low Energy 403

Provisionamento de Wi-Fi ... 404
Servidor de atributos genéricos ... 410
Bootloader para o Microchip Curiosity MZEF PIC32 ... 413
Estados de bootloader ... 413
Dispositivo flash ... 415
Estrutura de imagem da aplicação ... 415
Cabeçalho da imagem ... 416
Descritor da imagem ... 417
Trailer de imagem ... 418
Configuração de bootloader ... 419
Criação do bootloader ... 420
AWS IoT Device Defender ... 421
Introdução ... 421
Funcionalidade ... 421
AWS IoT Greengrass ... 426
Amazon EC2 ... 428
coreHTTP ... 430
Autenticação mútua da coreHTTP ... 431
Upload da coreHTTP do Amazon S3 ... 433
Download do coreHTTP do S3 ... 436
CoreHTTP com threads múltiplas ... 439
AWS IoT Empregos ... 444
Introdução ... 444
Organização de código-fonte ... 444
Configuração da conexão do agente MQTT do AWS IoT ... 445
Funcionalidade ... 445
coreMQTT ... 447
Autenticação mútua da coreMQTT ... 447
Compartilhamento de conexão coreMQTT ... 452
Over-the-air atualizações ... 458
Over-the-air configurações de demonstração ... 463
Texas Instruments CC322 0SF-LAUNCHXL ... 464
Microchip Curiosity PIC32 MZEF ... 466
Expressivo ESP32 ... 471
Renesas N RX65 ... 472
AWS IoT Device Shadow ... 497

Introdução 497
Funcionalidade 498
Conecte-se ao corretor AWS IoT MQTT 500
Exclusão de documento de sombra 500
Assinatura em tópicos de sombra 501
Envio de atualizações de sombras 501
Lidar com mensagens delta de sombra e mensagens de atualização de sombra 502
Secure Sockets 503
Usando AWS IoT Device Tester para FreeRTOS 505
Versões suportadas do IDT para FreeRTOS 507
Versão mais recente do IDT para FreeRTOS 507
Versões anteriores do IDT 508
Versões não compatíveis do IDT 510
Pré-requisitos 514
Fazer download do FreeRTOS 514
Qualificação LTS (qualificação para FreeRTOS que usam bibliotecas LTS) 515
Faça download do IDT para FreeRTOS 515
..... 516
AWS IoT Device Tester política gerenciada 517
(Opcional) Instale o AWS Command Line Interface 518
Preparação para testar sua placa de microcontrolador pela primeira vez 518
Adicionar camadas de transferência da biblioteca 519
Configure suas AWS credenciais 519
Crie um grupo de dispositivos no IDT para FreeRTOS 519
Configuração de parâmetros de compilação, atualização e teste 525
Execução de testes de Bluetooth Low Energy 538
Pré-requisitos 539
Configuração do Raspberry Pi 539
Configuração do dispositivo do FreeRTOS 543
Execução de testes de BLE 543
Solução de problemas de testes de BLE 544
Como executar o pacote de qualificação do FreeRTOS 544
Comandos do IDT para FreeRTOS 548
Teste para requalificação 550
Versões do pacote de testes 550
Noções básicas de resultados e logs 551

Visualização de resultados 552
Usar o IDT para desenvolver e executar os próprios conjuntos de testes 556
Baixe a versão mais recente do IDT para o FreeRTOS 557
Fluxo de trabalho de criação de pacotes de teste 557
Tutorial: compile e execute o pacote de teste de amostra do IDT 558
Tutorial: desenvolva um pacote de teste simples do IDT 564
Criação de arquivos de configuração do conjunto de testes do IDT 573
Configuração da máquina de estado do IDT 581
Criação de executáveis de casos de teste do IDT 605
Use o contexto do IDT 613
Definir configurações para os executores de teste 618
Depure e execute pacotes de teste personalizados 630
Analise os resultados e logs dos testes do IDT 634
Métricas de uso do IDT 640
Solução de problemas 647
Solução de problemas de configuração de dispositivos 648
Solução de erros de tempo limite 660
Recurso e AWS cobranças do celular 661
Política de geração de relatórios de qualificação 661
Política de suporte 661
Segurança em AWS 663
Gerenciamento de Identidade e Acesso 664
Público 664
Autenticação com identidades 665
Gerenciar o acesso usando políticas 669
Como o FreeRTOS funciona com o IAM 671
Exemplos de políticas baseadas em identidade 678
Solução de problemas 681
Validação de conformidade 683
Resiliência 685
Segurança da infraestrutura 685
..... dclxxxvii

O que é o FreeRTOS?

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 175 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/aws/amazon-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). A fonte de redução da documentação está disponível em https://github.com/awsdocs/aws-freertos-docs. Ela é liberada de acordo com a licença Creative Commons (CC BY-ND).

Download do código-fonte do FreeRTOS

Você pode clonar ou baixar FreeRTOS em. GitHub Consulte o arquivo README.md para obter instruções.

Versionamento do FreeRTOS

O kernel e os componentes do FreeRTOS são lançados individualmente e usam versionamento semântico. As versões do FreeRTOS integradas são feitas periodicamente. Todas as versões usam versionamento baseado em data com o formato AAAAMM.NN, em que:

- A representa o ano.
- M representa o mês.
- N representa a ordem de liberação dentro do mês designado (00 sendo a primeira versão).

Por exemplo, uma segunda versão em julho de 2021 seria 202107.01.

Anteriormente, as versões principais do FreeRTOS usavam versionamento semântico. Embora tenha mudado para o controle de versão baseado em datas (o FreeRTOS 1.4.8 foi atualizado para o FreeRTOS Reference AWS Integrations 201906.00), o kernel do FreeRTOS e cada biblioteca individual do FreeRTOS ainda mantêm o controle de versão semântico. No versionamento semântico, o próprio número da versão (X.Y.Z) indica se a versão é principal, secundária ou pontual. Você pode usar a versão semântica de uma biblioteca para avaliar o escopo e o impacto de uma nova versão em sua aplicação.

As versões LTS são mantidas de forma diferente de outros tipos de versão. Versões principais e secundárias são frequentemente atualizadas com novos recursos, além de resoluções de defeitos. As versões LTS são atualizadas somente com alterações para abordar defeitos críticos e vulnerabilidades de segurança. Nenhum novo recurso é introduzido em uma determinada versão LTS após o lançamento. Elas são mantidas por pelo menos três anos após o lançamento e oferecem aos fabricantes de dispositivos a opção de usar uma linha de base estável em vez de uma linha de base mais dinâmica representada por versões principais e secundárias.

Arquitetura do FreeRTOS

Normalmente, o FreeRTOS é atualizado para dispositivos como uma única imagem compilada com todos os componentes necessários para as aplicações do dispositivo. Essa imagem combina a funcionalidade dos aplicativos escritos pelo desenvolvedor incorporado, as bibliotecas de software fornecidas pela Amazon, o kernel do FreeRTOS e os pacotes de suporte de drivers e placas BSPs () para a plataforma de hardware. Independente do microcontrolador individual que está sendo usado, os desenvolvedores de aplicações incorporadas podem esperar as mesmas interfaces padronizadas para o kernel do FreeRTOS e todas as bibliotecas de software do FreeRTOS.

Plataformas de hardware qualificadas para o FreeRTOS

As seguintes plataformas de hardware estão qualificados para o FreeRTOS:

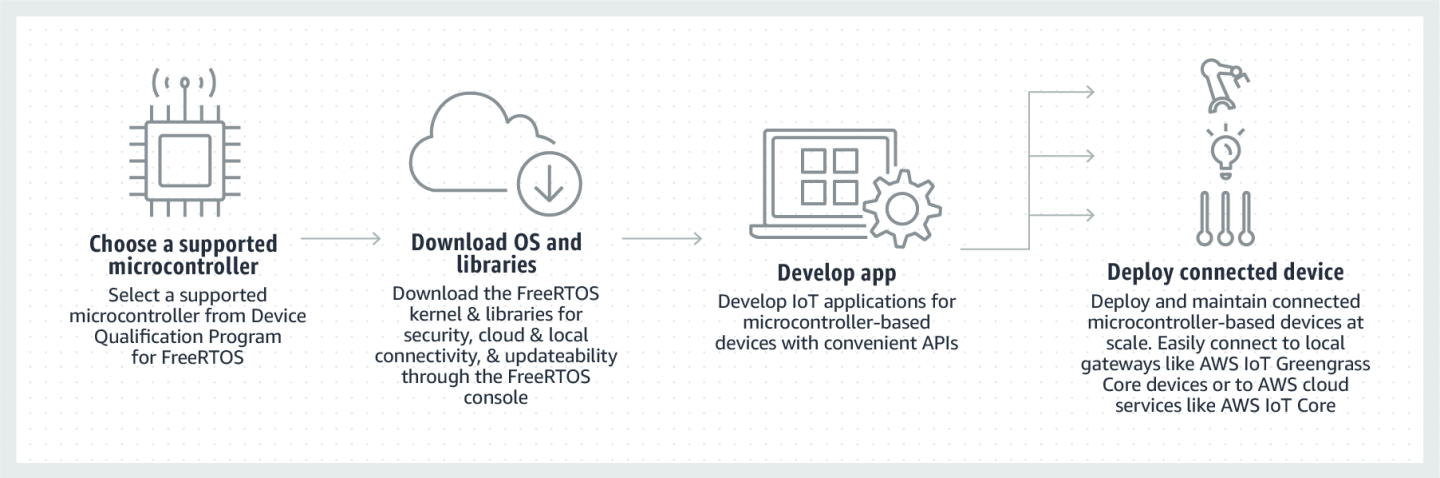
- ATECC6Kit de provisionamento 08A Zero Touch para AWS IoT
- Kit de desenvolvimento Cypress CYW9439 07 AEVAL1 F
- Kit de desenvolvimento Cypress CYW9549 07 AEVAL1 F
- Expressivo - C ESP32 DevKit
- Expressivo ESP-WROVER-KIT
- Kit de conectividade IoT Infineon XMC48 00
- Kit inicial Marvell MW32 0 AWS IoT
- Kit inicial da Marvell MW322 AWS IoT
- MediaTek MT7697Kit de desenvolvimento Hx
- Pacote Microchip Curiosity PIC32 MZEF
- Nórdico em RF5284 0-DK
- NuMaker-IoT-M487
- Módulo LPC54 IoT NXP 018
- Solução OPTIGA Trust X Security
- Módulo de IoT Renesas RX65 N RSK
- STMicroelectronicsSTM32Node IoT do Kit L4 Discovery
- Texas Instruments CC322 0SF-LAUNCHXL
- 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.

- Documentação do FreeRTOS adicional está disponível em freertos.org, incluindo o Manual de Referência do FreeRTOS v10.0.0.
- Para perguntas sobre o FreeRTOS para a equipe de engenharia do FreeRTOS, você pode abrir uma edição na página do FreeRTOS. GitHub
- Em caso de dúvidas técnicas sobre o FreeRTOS, visite os Fóruns da comunidade do FreeRTOS.
- Para obter mais informações sobre como conectar dispositivos a AWS IoT, consulte o Guia do AWS IoT Core desenvolvedor e o capítulo sobre provisionamento de dispositivos nesse guia.
- Para obter suporte técnico AWS, visite o AWS Support Center.
- Para perguntas sobre AWS cobrança, serviços de conta, eventos, abuso ou outros problemas com AWS, visite a página Fale conosco.

Princípios básicos do kernel do FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O kernel do FreeRTOS é um sistema operacional em tempo real compatível com inúmeras arquiteturas. É ideal para criar aplicações 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.

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. As tarefas bloqueadas não exigem manutenção periódica demorada. Direct-to-taskas notificações permitem a sinalização rápida de tarefas, praticamente sem sobrecarga de RAM. Eles podem ser usados na maioria dos cenários de intertarefas e interrupt-to-task sinalização.

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 maior parte da up-to-date documentação sobre o kernel do 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

Uma aplicação incorporada que usa um RTOS pode ser estruturada como um conjunto de tarefas independentes. Cada tarefa é executada em seu próprio contexto, sem dependência de outras tarefas. Somente uma tarefa na aplicação 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 da aplicação.

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 da aplicação 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 aplicações

Quando as aplicações precisam de memória, elas 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.

A fila APIs permite 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, direct-to-task como notificações e buffers de fluxo e mensagem, oferecem alternativas leves às filas em muitos cenários comuns de design.

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.

Direct-to-task notificações

As notificações de tarefas permitem que as tarefas interajam com outras tarefas e se sincronizem com as rotinas de interrupção do serviço (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 direct-to-task notificações. 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 direct-to-task notificações. 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`.

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 aplicações 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 aplicação 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 aplicação específicos com o arquivo de cabeçalho `FreeRTOSConfig.h`. Toda aplicação criada no kernel deve ter um arquivo de cabeçalho `FreeRTOSConfig.h` em seu caminho de inclusão do pré-processador. `FreeRTOSConfig.h` é específico da aplicação e deve ser colocado sob um diretório de aplicação 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.

AWS IoT SDK de dispositivo para C incorporado

Important

Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Note

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

O AWS IoT Device SDK para C incorporado (C-SDK) é uma coleção de arquivos de origem C sob a licença de código aberto do MIT que podem ser usados em aplicativos incorporados para conectar com segurança dispositivos de IoT a. AWS IoT Core Ele inclui um cliente MQTT, cliente HTTP, JSON Parser e AWS IoT Device Shadow, AWS IoT Jobs 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.

O AWS IoT Device SDK para C incorporado (C-SDK) é uma coleção de arquivos de origem C sob a licença de código aberto do MIT que podem ser usados em aplicativos incorporados para conectar com segurança dispositivos de IoT a. AWS IoT Core Ele inclui um cliente MQTT, cliente HTTP, JSON Parser e AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT Fleet Provisioning 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.

Geralmente, AWS IoT Device SDK para C incorporado é direcionado a dispositivos com recursos limitados que exigem um tempo de execução otimizado da linguagem C. Você pode usar o SDK em qualquer sistema operacional e hospedá-lo em qualquer tipo de processador (por exemplo, MCUs e MPUs). No entanto, se seus dispositivos tiverem memória e recursos de processamento suficientes, recomendamos que você use um AWS IoT dispositivo de ordem superior SDKs.

Para obter mais informações, consulte:

- AWS IoT SDK de dispositivo para C incorporado em GitHub

- AWS IoT SDK de dispositivo para Leiame C incorporado
- AWS IoT SDK de dispositivo para amostras C incorporadas

Conceitos básicos do FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 uma aplicação de demonstração simples em uma placa de microcontrolador qualificada.

Ao longo deste tutorial, presumimos que você esteja familiarizado com AWS IoT o AWS IoT console. Caso contrário, recomendamos que você conclua o tutorial de Conceitos básicos da AWS IoT antes de continuar.

Tópicos:

- Aplicação de demonstração do FreeRTOS
- Primeiras etapas
- Solução de problemas de conceitos básicos
- Usando CMake com FreeRTOS
- Provisionamento de chaves no modo de desenvolvedor
- Guias de conceitos básicos específicos da placa

Aplicação de demonstração do FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O aplicativo de demonstração neste tutorial é a demonstração da Autenticação Mútua CoreMQTT definida no arquivo. *freertos*/demos/coreMQTT/mqtt_demo_mutual_auth.c Ele usa o Biblioteca coreMQTT para se conectar à AWS nuvem e, em seguida, publicar mensagens periodicamente em um tópico do MQTT hospedado pelo corretor do AWS IoT MQTT.

Apenas uma aplicação de demonstração do FreeRTOS pode ser executada 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 a aplicação em execução. Para este tutorial, você não precisa habilitar ou desabilitar as demonstrações. A demonstração da Autenticação Mútua do CoreMQTT está ativada por padrão.

Para obter mais informações sobre as aplicações de demonstração incluídos com o FreeRTOS, consulte Demonstrações do FreeRTOS.

Primeiras etapas

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Para começar a usar o FreeRTOS AWS IoT com, você deve ter AWS uma conta, um usuário do IAM com permissão para acessar AWS IoT e os 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 ESP32 - DevKit C,, ou o ESP32 -WROOM-32SE ESP-WROVER-KIT, pule essas etapas e vá para. Começando com o Espressif ESP32 - DevKit C e o ESP-WROVER-KIT
> - Se você estiver usando o Nordic n RF5284 0-DK, pule essas etapas e vá para. Começando com o Nordic n 0-DK RF5284

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 a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma ligação ou mensagem de texto 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 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, você pode visualizar a atividade atual da sua conta e gerenciar sua conta acessando https://aws.amazon.com/e escolhendo 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 a 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. Habilita o Centro de Identidade do IAM.

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

2. No Centro de Identidade do IAM, conceda o acesso administrativo a 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 o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

  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 Criar um conjunto de permissões 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 Adicionar grupos 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.

Depois de configurar sua AWS conta e permissões, continueRegistrando sua placa MCU com AWS IoT.

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
   ```

   O resultado deve ser semelhante ao seguinte:

   ```
   {
       "endpointAddress": "xxxxxxxxxxxxx.iot.us-west-2.amazonaws.com"
   }
   ```

   Neste exemplo, a região é `us-west-2`.
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 sua diretoria for registrada AWS IoT, você poderá continuarFazer download do FreeRTOS.

Fazer download do FreeRTOS

Você pode baixar o FreeRTOS do repositório do FreeRTOS. GitHub

> **Note**
>
> Se você está começando com os kits de desenvolvimento Cypress CYW9549 07 AEVAL1 F ou CYW9439 07 AEVAL1 F, você deve baixar os FreeRTOS em. GitHub Consulte o arquivo README.md para obter instruções. No momento, as configurações do FreeRTOS para essas placas não estão disponíveis no console 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, selecione Configurações.

   Seu AWS IoT endpoint é exibido no 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` *`The security type of your Wi-Fi network`*

  Os tipos de segurança válidos são:

  - `eWiFiSecurityOpen` (Aberto, sem segurança)
  - `eWiFiSecurityWEP` (segurança WEP)
  - `eWiFiSecurityWPA` (segurança WPA)
  - `eWiFiSecurityWPA2`(WPA2 segurança)

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 os certificados do dispositivo do AWS IoT console, mas não pode recuperar as 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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 aparece no AWS IoT console depois que eu executo o projeto de demonstração Hello World. 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	nenhuma

Configuração do terminal	Valor
Parar	1 bit
Controle de fluxo	nenhuma

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.*
```

Usando CMake com FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Você pode usar CMake para gerar arquivos de construção do projeto a partir do código-fonte do aplicativo FreeRTOS e para criar 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 preferir trabalhar sem um IDE, você pode usar outras ferramentas de edição e depuração de código de terceiros para desenvolver e depurar seu código e, em seguida, usá-las CMake para criar e executar os aplicativos.

As seguintes placas suportam CMake:

- Expressivo - C ESP32 DevKit
- Expressivo ESP-WROVER-KIT
- Kit de conectividade IoT Infineon XMC48 00
- Kit inicial Marvell MW32 0 AWS IoT
- Kit inicial da Marvell MW322 AWS IoT
- Pacote Microchip Curiosity PIC32 MZEF
- Kit de desenvolvimento Nordic n RF5284 10 DK
- STMicroelectronicsSTM32Node IoT do Kit L4 Discovery
- Texas Instruments CC322 0SF-LAUNCHXL
- Simulador do Microsoft Windows

Consulte os tópicos abaixo 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
- Criando FreeRTOS com 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. CMake suporta todas as versões do Windows, macOS e Linux

O subsistema do Windows para Linux (WSL) não é compatível. Use nativo CMake em máquinas Windows.

- Você deve ter a CMake versão 3.13 ou superior instalada.

  Você pode baixar a distribuição binária CMake de em CMake.org.

  > **Note**
  >
  > Se você baixar a distribuição binária de CMake, certifique-se de adicionar o CMake executável à variável de ambiente PATH antes de usar a CMake partir da linha de comando.

  Você também pode baixar e instalar CMake usando um gerenciador de pacotes, como homebrew no macOS e scoop ou chocolatey no Windows.

  > **Note**
  >
  > As versões de CMake pacotes fornecidas nos gerenciadores de pacotes para muitas distribuições Linux são out-of-date. Se o gerenciador de pacotes da sua distribuição não fornecer a versão mais recente do CMake, você pode tentar gerenciadores de pacotes alternativos, como `linuxbrew` ou `unix`.

- Você deve ter um sistema de compilação nativo compatível.

  CMake pode ter como alvo muitos sistemas de compilação nativos, incluindo GNU Make ou 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 terminar de desenvolver seu aplicativo, você pode invocar a ferramenta de CMake linha de comando para criar seu projeto a partir do VS Code. Para obter mais informações sobre como usar CMake para criar aplicativos FreeRTOS, consulte. Criando FreeRTOS com 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"
    }
]
```

Criando FreeRTOS com CMake

CMake por padrão, o sistema operacional host é o sistema de destino. Para usá-lo para compilação cruzada, é CMake necessário 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 forma de fornecer esse arquivo CMake depende se você está usando a interface de linha de CMake comando ou a GUI. Para obter mais detalhes, siga as instruções de Geração de arquivos de compilação (CMake ferramenta de linha de comando) abaixo. Para obter mais informações sobre compilação cruzada CMake, consulte CrossCompilingno wiki oficial CMake .

Para criar um projeto CMake baseado

1. Execute CMake para gerar os arquivos de compilação para um sistema de compilação nativo, como Make ou Ninja.

   Você pode usar a ferramenta de CMake linha de comando ou a CMake GUI 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 (CMake ferramenta de linha de comando) e Gerando arquivos de compilação (CMake GUI).
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 (CMake ferramenta de linha de comando)

Você pode usar a ferramenta de CMake linha de comando (cmake) para gerar arquivos de compilação para 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 o CC322 0 Launchpad e o compilador for GCC for ARM, você poderá emitir o seguinte comando para criar os arquivos de origem do diretório atual em um diretório chamado: *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 porque 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 do conjunto de ferramentas não usar caminhos absolutos para o compilador e você não tiver adicionado o compilador à variável de PATH ambiente, CMake talvez não consiga encontrá-lo. Para garantir que CMake encontre seu arquivo do conjunto de ferramentas, você pode usar a AFR_TOOLCHAIN_PATH opção. 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, CMake adiciona sinalizadores de depuração às opções de compilação e cria FreerTOS com símbolos de depuração.

```
# Build with debug symbols
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -DCMAKE_BUILD_TYPE=debug -S . -B build-directory
```

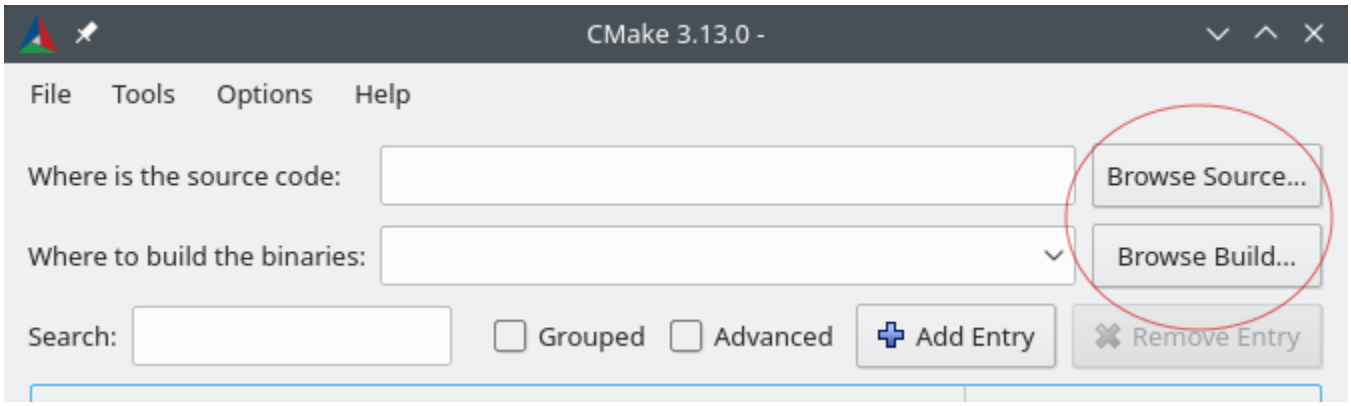
Você também pode definir o CMAKE_BUILD_TYPE como release para adicionar sinalizadores de otimização às opções de compilação.

Gerando arquivos de compilação (CMake GUI)

Você pode usar a CMake GUI para gerar arquivos de compilação do FreeRTOS.

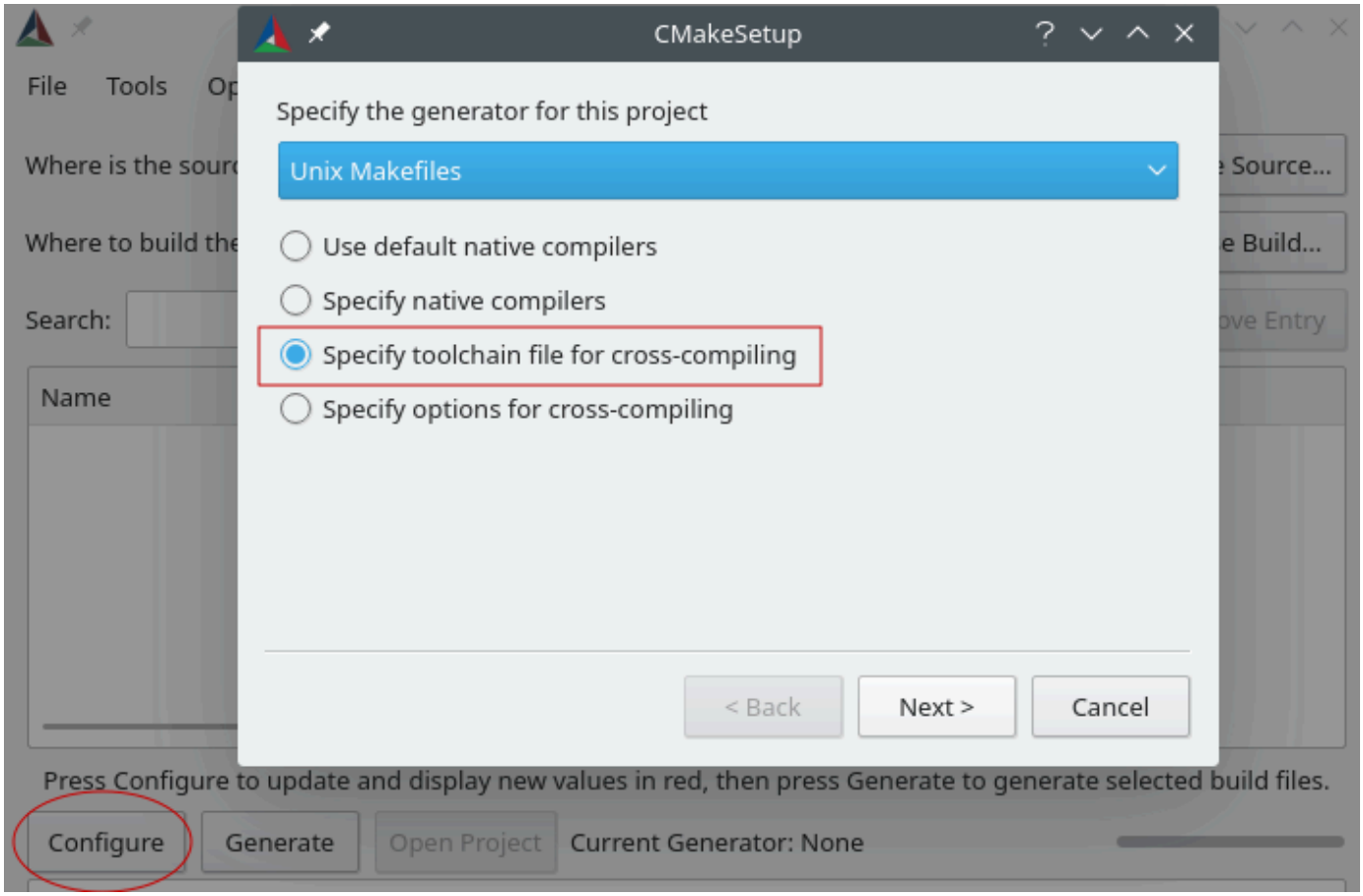
Para gerar arquivos de compilação com a CMake GUI

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. Nesse caso, exclua o CMake cache escolhendo Excluir cache no menu 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.

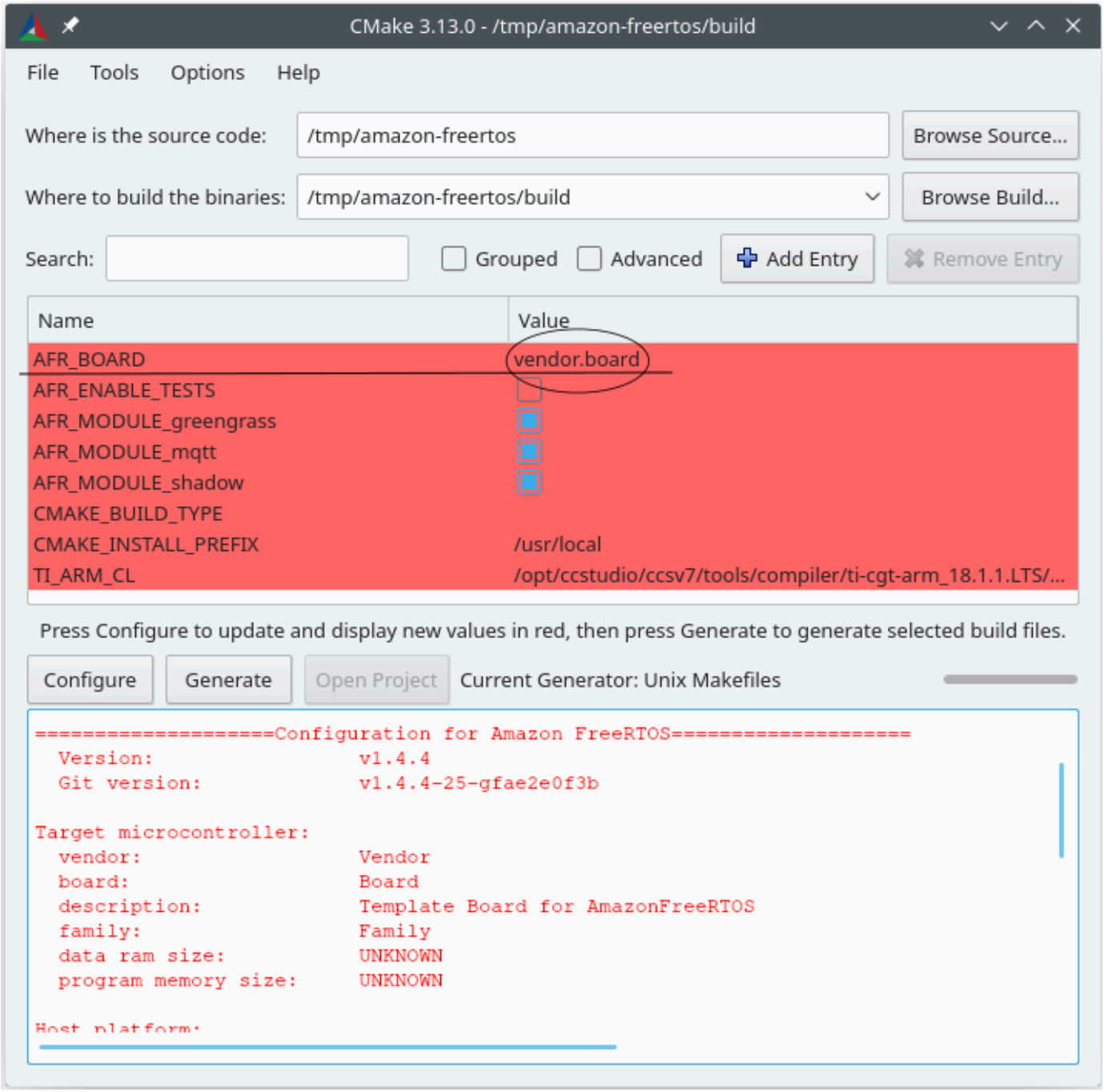
   A configuração padrão do FreeRTOS é a placa de modelo, que não fornece nenhum destino de camada portátil. Como resultado, uma janela será exibida com a mensagem Error in configuration process.

> **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 `AFR_TOOLCHAIN_PATH` variável na GUI para saber 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. 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
```

Edifício com CMake

Você também pode usar a ferramenta de CMake linha de comando para criar Freertos. CMake fornece uma camada de abstração para chamar sistemas de compilação nativos. Por exemplo:

```
cmake --build build_dir
```

Aqui estão alguns outros usos comuns do modo de construção da ferramenta CMake de linha de comando:

```
# 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 CMake compilação, consulte a CMake documentação.

Provisionamento de chaves no modo de desenvolvedor

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 chave privada de 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 emConfiguração das demonstrações do FreeRTOS, mas pule a última etapa (ou seja, não faça Para formatar suas AWS IoT credenciais). 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 de série.

```
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 for criada e registrada AWS IoT, use o comando a seguir para emitir um certificado de cliente com base na CSR do dispositivo que foi 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. Você também precisará importar seu certificado de Autoridade Certificadora (CA), pois ele é necessário para que a primeira autenticação seja bem-sucedida AWS IoT ao usar o 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 AWS IoT registro conforme descrito em Use seu próprio certificado. Você deve criar uma AWS IoT coisa nova, anexar o certificado PENDENTE e uma política à sua coisa e, em seguida, marcar o certificado como ATIVO. Todas essas etapas podem ser executadas manualmente no AWS IoT console.

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 corretor AWS IoT MQTT será bem-sucedida.

Guias de conceitos básicos específicos da placa

Important

Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Depois de concluir Primeiras etapas, consulte o guia da placa para obter instruções específicas sobre como começar a usar o FreeRTOS:

- Começando com o kit de desenvolvimento Cypress CYW9439 07 AEVAL1 F
- Começando com o kit de desenvolvimento Cypress CYW9549 07 AEVAL1 F
- Introdução ao kit Cypress CY8 CKIT-064S0S2-4343W
- Introdução ao kit de conectividade IoT Infineon XMC48 00
- Introdução ao MW32x AWS IoT Starter Kit
- Introdução ao kit de desenvolvimento MediaTek MT7697 Hx
- Começando com o Microchip Curiosity PIC32 MZ EF
- Começando com o Nuvoton 487 NuMaker-IoT-M
- Introdução ao módulo de LPC54 IoT NXP 018
- Começando com o Renesas Starter Kit+ para N-2MB RX65
- Introdução ao STMicroelectronics STM32 L4 Discovery Kit IoT Node

- Começando com a Texas Instruments CC322 0SF-LAUNCHXL
- Conceitos básicos do simulador de dispositivo do Windows
- Começando com o kit de IoT industrial Xilinx Avnet MicroZed

> **Note**
>
> Você não precisa concluir Primeiras etapas para os seguintes guias independentes de Conceitos básicos do FreeRTOS:
>
> - Introdução ao simulador Microchip ATECC6 08A Secure Element com Windows
> - Começando com o Espressif ESP32 - DevKit C e o ESP-WROVER-KIT
> - Introdução ao ESP32 Espressif -WROOM-32SE
> - Introdução ao kit de conectividade XMC48 IoT Infineon OPTIGA Trust X e 00
> - Começando com o Nordic n 0-DK RF5284

Começando com o kit de desenvolvimento Cypress CYW9439 07 AEVAL1 F

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Cypress CYW9439 07 AEVAL1 F Development Kit. Se você não tiver o kit de desenvolvimento Cypress CYW9439 07 AEVAL1 F, visite o Catálogo de dispositivos do AWS parceiro para comprar um de 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 AWS IoT e fazer o download dos FreeRTOS para conectar seu dispositivo à nuvem. 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.
- Conforme observado emFazer download do FreeRTOS, as portas FreeRTOS para Cypress estão atualmente disponíveis 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 uma aplicação 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 Avançado, selecione 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.

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.

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***`your-thing-name`*`/example/topic`** e selecione Inscreva-se no tópico.

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.

Começando com o kit de desenvolvimento Cypress CYW9549 07 AEVAL1 F

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Cypress CYW9549 07 AEVAL1 F Development Kit. Se você não tiver o kit de desenvolvimento Cypress CYW9549 07 AEVAL1 F, visite o Catálogo de dispositivos AWS parceiros para comprar um de 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 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`*.

> **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.
> - Conforme observado emFazer download do FreeRTOS, as portas FreeRTOS para Cypress estão atualmente disponíveis 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 uma aplicação 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 Avançado, selecione 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.

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.

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*/example/topic** e selecione Inscreva-se no tópico.

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.

Introdução ao kit Cypress CY8 CKIT-064S0S2-4343W

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 AWS IoT os FreeRTOS para conectar seu dispositivo à nuvem. AWS Para obter instruções, consulte Primeiras etapas. Depois de concluir os pré-requisitos, você terá um pacote FreeRTOS com credenciais. 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 CMake um fluxo de construção ou com o Make. Você pode usar ModusToolbox para seu fluxo de construção do Make. Você pode usar o IDE Eclipse fornecido com ModusToolbox ou um IDE parceiro, como IAR EW-ARM, Arm MDK ou Microsoft Visual Studio Code. O IDE do Eclipse é compatível com os sistemas operacionais Windows, macOS e Linux.

Antes de começar, baixe e instale o ModusToolbox software mais recente. Para obter mais informações, consulte o Guia ModusToolbox de instalação.

Ferramentas de atualização para ModusToolbox 2.1 ou versões anteriores

Se você estiver usando o IDE 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 Aplicações 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. Baixe o arquivo do 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. Baixe o arquivo do 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 CMake para gerar arquivos de compilação do projeto a partir do código-fonte do aplicativo FreeRTOS, criar o projeto usando sua ferramenta de compilação preferida e, em seguida, programar o kit usando o OpenOCD. Se você preferir usar uma ferramenta de GUI para programar com o CMake fluxo, baixe e instale o Cypress Programmer na página da Cypress Programming Solutions. Para obter mais informações, consulte Usando CMake com FreeRTOS.

Configuração do hardware

Siga estas etapas para configurar o hardware do kit.

1. Provisionamento do kit

   Siga o Guia de Provisionamento das instruções do kit CY8 CKIT-064S0S2-4343W para provisionar seu kit com segurança para. AWS IoT

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. Certifique-se de seguir as etapas no Guia de aprovisionamento do kit CY8 CKIT-064S0S2-4343W.
2. Compile a demonstração do FreeRTOS.

   a. Abra o Eclipse IDE 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 a aplicação fazendo uma das seguintes etapas:

- No Painel rápido, selecione Compilar aplicação aws_demos.
- Escolha Projeto e escolha Compilar tudo.

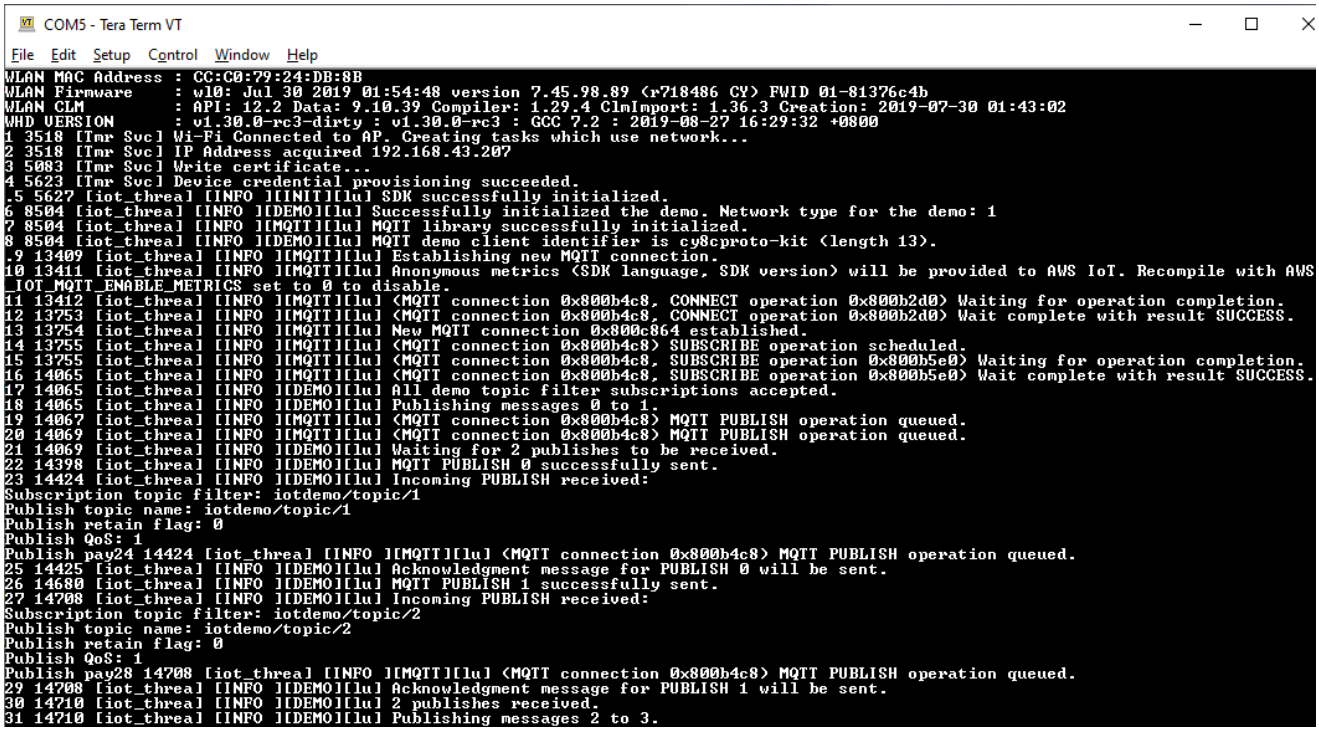
Verifique se o projeto é compilado sem erros.

3. 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 a aplicação de demonstração começa a ser executada 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. 1 a 4 — O dispositivo se conecta ao ponto de acesso (AP) configurado e é provisionado conectando-se ao AWS servidor 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.

4. Usando CMake com FreeRTOS

Você também pode usar CMake para criar e executar o aplicativo de demonstração. Para configurar um sistema CMake de compilação nativo, consultePré-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 `-G` opção porque 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 não `arm-none-eabi-gcc` estiver no caminho do shell, você também precisará definir a `AFR_TOOLCHAIN_PATH` CMake variável.

Example

```
-DAFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

b. Use o comando a seguir para criar o projeto usando CMake.

```
cmake --build build_dir
```

c. Por fim, programe os arquivos `cm0.hex` e `cm4.hex` gerados em *`build_dir`* usando o Cypress Programmer.

5. Monitoramento de mensagens MQTT na Nuvem

   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, siga estas etapas.

   a. Faça login no console do AWS IoT.

   b. No painel de navegação, escolha Test (Teste) para abrir o cliente MQTT.

   c. Em Tópico de assinatura, insira ***`your-thing-name`*`/example/topic`** e selecione Assinar o tópico.

   d. Reinicie o kit para forçá-lo a publicar mensagens MQTT, que podem ser vistas no cliente de teste do console.

Execução de outras demonstrações

As aplicações de demonstração a seguir foram testadas e verificadas 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
- Over-the-Air Demo de atualizações
- Demonstração do cliente Echo de Secure Sockets
- AWS IoT Demonstração do Device Shadow

Depuração

Os KitProg 3 do kit oferecem suporte à depuração pelo protocolo SWD.

- Para depurar o aplicativo FreeRTOS, selecione o projeto aws_demos no espaço de trabalho e, em seguida, selecione aws_demos Debug (3) no Painel rápido. KitProg

Atualizações OTA

PSoO C 64 MCUs passou em todos os testes de qualificação exigidos do FreeRTOS. No entanto, o recurso opcional over-the-air (OTA) implementado na biblioteca de AWS firmware PSo C 64 Standard Secure ainda está pendente de avaliação. O recurso 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 sucesso é aplicada a um dispositivo usando o PSo C64 Standard Secure — AWS MCU e o dispositivo não consegue se comunicar AWS IoT Core, o dispositivo não pode voltar automaticamente para a imagem original em boas condições. Isso pode fazer com que o dispositivo fique inacessível AWS IoT Core para futuras atualizações. Essa funcionalidade ainda está sendo desenvolvida pela equipe do Cypress.

Para obter mais informações, consulte Atualizações OTA AWS e o kit CY8 CKIT-064S0S2-4343W. Caso existam mais dúvidas ou você precise de suporte técnico, entre em contato com a Comunidade de desenvolvedores da Cypress.

Introdução ao simulador Microchip ATECC6 08A Secure Element com Windows

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Microchip ATECC6 08A Secure Element com o Windows Simulator.

Você precisa do seguinte hardware:

- Quadro de cliques de elementos seguros Microchip ATECC6 08A
- SAMD21 XPlained Profissional
- 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 Neste tutorial, o caminho para o diretório de download do FreeRTOS é chamado *`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.

Configure o hardware Microchip ATECC6 08A

Antes de interagir com seu dispositivo Microchip ATECC6 08A, você deve primeiro programar o. SAMD21

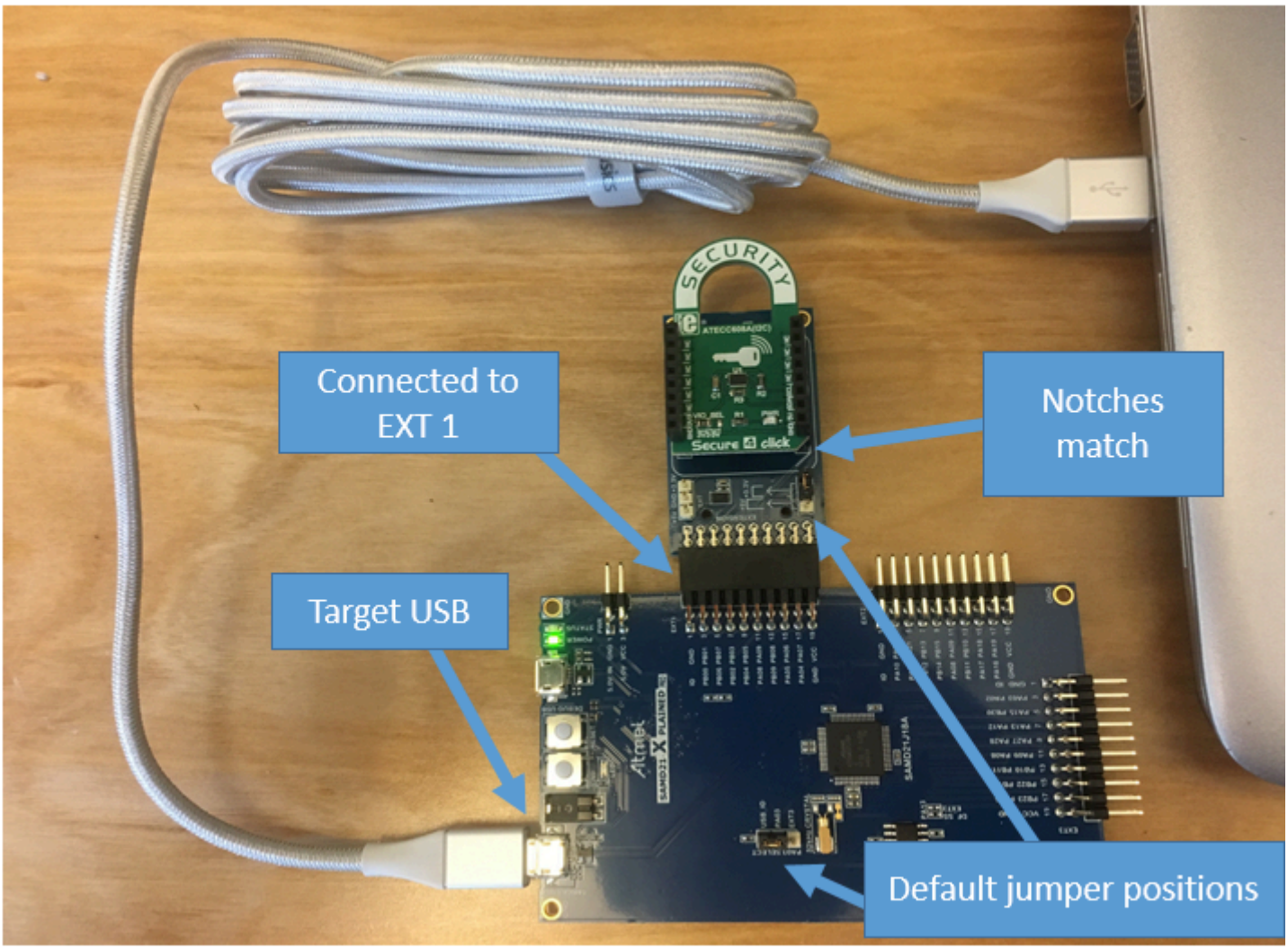
Para configurar a placa SAMD21 XPlained Pro

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 XPlained mikroBus Pro na placa no SAMD21 local. EXT1
3. Conecte a placa ATECC6 08A Secure 4 Click ao adaptador mikroBusX Pro XPlained . 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 a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma ligação ou mensagem de texto 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 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, você pode visualizar a atividade atual da sua conta e gerenciar sua conta acessando https://aws.amazon.com/e escolhendo 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 a 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. Habilita o Centro de Identidade do IAM.

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

2. No Centro de Identidade do IAM, conceda o acesso administrativo a 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 o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

  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 Criar um conjunto de permissões 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 Adicionar grupos 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. Para baixar os FreeRTOS em: GitHub
   1. Navegue até o repositório do FreeRTOS GitHub .
   2. Escolha Código e, em seguida, Clonar ou baixar ZIP.
   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.

   4. No diretório *freertos*, confira a ramificação a ser usada.
2. Configure o ambiente de desenvolvimento.

a. Instale a versão mais recente do Win PCap.

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 ATECC6 08A 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 obter mais informações, consulte Configuração da interface de rede.

4. Provisione o Microchip ATECC6 08A.

   A Microchip forneceu várias ferramentas de script para ajudar na configuração das peças ATECC6 08A. 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. Criar e registrar uma autoridade de certificação com a AWS.
   2. Gere suas chaves no Microchip ATECC6 08A e exporte a chave pública e o número de série do dispositivo.
   3. Gerar um certificado para o dispositivo e registrar esse certificado na 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.

Começando com o Espressif ESP32 - DevKit C e o ESP-WROVER-KIT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Siga este tutorial para começar a usar o Espressif ESP32 - DevKit C equipado com os módulos ESP32 -WROOM-32, -SOLO-1 ou ESP-WROVER e ESP32 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-SOZINO-1
- ESP32-KIT WROVER

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-KITv4.1 no site da Espressif.

> **Note**
>
> Atualmente, a porta FreeRTOS ESP32 para -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 a placa do microcontrolador
3. Fazer a compilação cruzada de um aplicativo de demonstração do FreeRTOS 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 a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma ligação ou mensagem de texto 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 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, você pode visualizar a atividade atual da sua conta e gerenciar sua conta acessando https://aws.amazon.com/e escolhendo 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 a 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. Habilita o Centro de Identidade do IAM.

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

2. No Centro de Identidade do IAM, conceda o acesso administrativo a 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 o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

  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 Criar um conjunto de permissões 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 Adicionar grupos 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.

> **Note**
>
> O FreeRTOS versão 202012.00 oferece suporte ao ESP-IDF v3.3. Se você estiver usando uma versão posterior do FreeRTOS, siga as instruções específicas IDFv4 de .2 em. Conceitos básicos do ESP-IDF v4.2

Conceitos básicos do ESP-IDF v3.3

Esta seção mostra como usar o ESP-IDF v3.3 no hardware da Espressif. Para usar o ESP-IDF v4.2, consulte Conceitos básicos do ESP-IDF v4.2.

Configuração do hardware Espressif

Para obter mais informações sobre como configurar o hardware da placa de desenvolvimento ESP32 - DevKit C, consulte o Guia de introdução ESP32 - DevKit C V4.

Para obter informações sobre como configurar o hardware da placa de ESP-WROVER-KIT desenvolvimento, consulte o Guia de introdução da ESP-WROVER-KIT V4.1.

> **Important**
>
> Quando chegar à seção Get Started dos guias do Espressif, siga as etapas abaixo.

Configuração do ambiente de desenvolvimento

Para se comunicar com sua placa, você deve baixar e instalar uma cadeia de ferramentas.

Configuração da cadeia de ferramentas

> **Note**
>
> O ESP-IDF v3.3 não é compatível com a versão mais recente do compilador. ESP32 Se você já instalou a versão mais recente do ESP32 compilador, desinstale-a e use a versão compatível do compilador incluída na instalação do conjunto de ferramentas. Veja os links nesta seção. Para verificar a versão do compilador, execute o comando a seguir.
>
> ```
> xtensa-esp32-elf-gcc --version
> ```

Para configurar a cadeia de ferramentas, siga estas instruções do sistema operacional de sua máquina host:

- Configuração padrão da cadeia de ferramentas (ESP-IDF v3.3) para Windows
- Configuração padrão da cadeia de ferramentas (ESP-IDF v3.3) para macOS
- Configuração padrão da cadeia de ferramentas (ESP-IDF v3.3) para Linux

> **Important**
>
> Quando você chegar nas instruções de "Obter ESP-IDF" em Próximas etapas, pare e retorne para as instruções desta página.
> Se você seguiu as instruções de “Get ESP-IDF” em Next Steps, agora ou em uma ocasião anterior, você deve limpar a variável de ambiente `IDF_PATH` do sistema antes de continuar.

Se você seguiu as instruções de “Get ESP-IDF”, essa variável de ambiente é definida automaticamente.

Instalar CMake

O sistema de CMake compilação é necessário para criar os aplicativos de demonstração e teste do FreeRTOS para esse dispositivo. O FreeRTOS oferece suporte às versões 3.13 ou posteriores.

Você pode baixar a versão mais recente CMake do em CMake.org. As distribuições binárias e de origem estão disponíveis.

Para obter mais detalhes sobre o uso CMake com FreeRTOS, consulte. Usando CMake com FreeRTOS

Estabelecimento de uma conexão serial

Para estabelecer uma conexão serial entre sua máquina host e o ESP32 - DevKit C, você deve instalar os drivers CP21 0x 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, você deve instalar alguns drivers de porta COM virtual FTDI. Você pode fazer download desses driver no FTDI.

Para obter mais informações, consulte Estabelecer conexão serial com ESP32. Depois de estabelecer uma conexão serial, anote a porta serial da conexão de sua placa. Você precisará dela durante a compilação da demonstração.

Faça download e configure os FreeRTOS

Depois de configurar seu ambiente, você pode baixar os Freertos GitHubem .Para obter instruções, consulte o arquivo README.md no site. GitHub

Configuração das aplicações de demonstração do FreeRTOS

1. Se você estiver executando macOS ou Linux, abra um prompt de terminal. Se você estiver executando o Windows, abra `mingw32.exe`. MinGW é um ambiente de desenvolvimento minimalista para aplicativos nativos do Microsoft Windows.
2. Para verificar se você tem Python 2.7.10 ou posterior instalado, execute

```
python --version
```

A versão instalada é exibida. Se você não tiver o Python 2.7.10 ou posterior instalado, poderá instalá-lo no site do Python.

3. Você precisa do AWS Command Line Interface (AWS CLI) para executar AWS IoT comandos. Se você estiver executando o Windows, use o easy_install awscli para instalar o AWS CLI no ambiente mingw32.

   Se você estiver executando o macOS ou o Linux, consulte Installing the AWS CLI.
4. Execute aws configure e configure o AWS CLI com seu ID de chave de AWS acesso, chave de acesso secreta e AWS região. Para obter mais informações, consulte Configurar a AWS CLI.
5. Use o comando a seguir para instalar o AWS SDK para Python (boto3):

   - No Windows, no ambiente mingw32, 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 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`(WPA2 segurança)

Para executar o script de configuração

1. Se você estiver executando macOS ou Linux, abra um prompt de terminal. Se você estiver executando o Windows, abra `mingw32.exe`.
2. 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 da IoT
- Anexa a política da IoT ao certificado e o certificado à coisa da AWS IoT
- Preenche o `aws_clientcredential.h` arquivo com seu AWS IoT endpoint, SSID Wi-Fi e credenciais
- 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`.

Compilar, instalar e executar o projeto de demonstração do FreeRTOS

Você pode usar CMake para gerar os arquivos de compilação, Make para criar o binário do aplicativo e o utilitário IDF do Espressif para atualizar sua placa.

Compilar o FreeRTOS no Linux e macOS (ESP-IDF v3.3)

Se você estiver usando o Windows, consulte Compilar o FreeRTOS no Windows (ESP-IDF v3.3).

Use CMake para gerar os arquivos de compilação e, em seguida, use Make para criar o aplicativo.

Para gerar os arquivos de compilação do aplicativo de demonstração com CMake

1. Altere os diretórios para a raiz do diretório de download do FreeRTOS.
2. Use o comando a seguir para gerar os arquivos de compilação.

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B your-build-directory
```

> **Note**
>
> Se você deseja compilar o aplicativo para depuração, adicione o sinalizador `-DCMAKE_BUILD_TYPE=Debug` a esse comando.
> Se você deseja gerar os arquivos de compilação do aplicativo de teste, adicione o sinalizador `-DAFR_ENABLE_TESTS=1`.
> O código fornecido pelo Espressif usa a pilha de IP leve (lwIP) como a pilha de rede padrão. Para usar a pilha de rede FreeRTOS+TCP em vez disso, adicione o sinalizador ao comando. `–DAFR_ESP_FREERTOS_TCP` CMake
> Para adicionar a dependência do LWIP para código não fornecido pelo fornecedor, adicione as linhas a seguir ao arquivo `CMakeLists.txt` de dependência,, para seu CMake componente Wi-Fi personalizado.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

Para criar o aplicativo com Make

1. Altere os diretórios para o diretório `build`.
2. Use o seguinte comando para criar o aplicativo com o Make.

```
make all -j4
```

> **Note**
>
> Você deve gerar os arquivos de compilação com o comando cmake toda vez que alternar entre o projeto `aws_demos` e o projeto `aws_tests`.

Compilar o FreeRTOS no Windows (ESP-IDF v3.3)

No Windows, você deve especificar um gerador de compilação para CMake. Caso contrário, o CMake padrão é Visual Studio. O Espressif recomenda oficialmente o sistema do build Ninja, porque ele funciona no Windows, no Linux e no MacOS. Você deve executar CMake comandos em um ambiente nativo do Windows, como cmd ou PowerShell. A execução de CMake comandos em um ambiente Linux virtual, como MSYS2 ou WSL, não é suportada.

Use CMake para gerar os arquivos de compilação e, em seguida, use Make para criar o aplicativo.

Para gerar os arquivos de compilação do aplicativo de demonstração com CMake

1. Altere os diretórios para a raiz do diretório de download do FreeRTOS.
2. Use o comando a seguir para gerar os arquivos de compilação.

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -GNinja -
S . -B build-directory
```

> **Note**
>
> Se você deseja compilar o aplicativo para depuração, adicione o sinalizador `-DCMAKE_BUILD_TYPE=Debug` a esse comando.
> Se você deseja gerar os arquivos de compilação do aplicativo de teste, adicione o sinalizador `-DAFR_ENABLE_TESTS=1`.

O código fornecido pelo Espressif usa a pilha de IP leve (lwIP) como a pilha de rede padrão. Para usar a pilha de rede FreeRTOS+TCP em vez disso, adicione o sinalizador ao comando. `-DAFR_ESP_FREERTOS_TCP` CMake
Para adicionar a dependência do LWIP para código não fornecido pelo fornecedor, adicione as linhas a seguir ao arquivo `CMakeLists.txt` de dependência,, para seu CMake componente Wi-Fi personalizado.

```
# Add a dependency on the bluetooth espressif component to the common
 component
set(COMPONENT_REQUIRES lwip)
```

Para o build do aplicativo

1. Altere os diretórios para o diretório `build`.
2. Invoque o Ninja para criar o aplicativo.

```
ninja
```

Ou use a CMake interface genérica para criar o aplicativo.

```
cmake --build build-directory
```

> **Note**
>
> Você deve gerar os arquivos de compilação com o comando cmake toda vez que alternar entre o projeto `aws_demos` e o projeto `aws_tests`.

Instalar e executar o FreeRTOS (ESP-IDF v3.3)

Esta seção mostra como usar o ESP-IDF v3.3 no hardware da Espressif. Para usar o ESP-IDF v4.2, consulte Conceitos básicos do ESP-IDF v4.2.

Use o utilitário IDF da Espressif (*freertos*`/vendors/espressif/esp-idf/tools/idf.py`) para atualizar sua placa, executar o aplicativo e visualizar os logs.

Para apagar a atualização da placa, acesse o diretório *freertos* e use o seguinte comando.

```
./vendors/espressif/esp-idf/tools/idf.py erase_flash -B build-directory
```

Para atualizar o binário do aplicativo para sua placa, use `make`

```
make flash
```

Você também pode usar o script do IDF para atualizar sua placa:

```
./vendors/espressif/esp-idf/tools/idf.py flash -B build-directory
```

Para monitorar:

```
./vendors/espressif/esp-idf/tools/idf.py monitor -p /dev/ttyUSB1 -B build-directory
```

> **Note**
>
> Você pode combinar esses comandos. Por exemplo:
>
> ```
> ./vendors/espressif/esp-idf/tools/idf.py erase_flash flash monitor -p /dev/
> ttyUSB1 -B build-directory
> ```

Conceitos básicos do ESP-IDF v4.2

Esta seção mostra como usar o ESP-IDF v4.2 no hardware da Espressif. Para usar o ESP-IDF v3.3, consulte Conceitos básicos do ESP-IDF v3.3.

> **Note**
>
> Os comandos do Linux neste tutorial exigem o uso do shell Bash.

Configuração do hardware Espressif

Para obter informações sobre como configurar o hardware da placa de desenvolvimento ESP32 - DevKit C, consulte o Guia de introdução ESP32 - DevKit C V4.

Para obter informações sobre como configurar o hardware da placa de ESP-WROVER-KIT desenvolvimento, consulte o Guia de introdução da 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.

Configuração do ambiente de desenvolvimento

Para se comunicar com sua placa, você deve baixar e instalar uma cadeia de ferramentas.

Configuração da cadeia de ferramentas

Para configurar a cadeia de ferramentas, siga estas instruções do sistema operacional de sua máquina host:

- Configuração padrão do conjunto de ferramentas e pré-requisitos (ESP-IDF v4.2) para Windows
- Configuração padrão do conjunto de ferramentas e pré-requisitos (ESP-IDF v4.2) para macOS
- Configuração padrão do conjunto de ferramentas e pré-requisitos (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.

Concluir a instalação (Linux/macOS)

O instalador do ESP-IDF para Windows instala todas as ferramentas necessárias. As plataformas Linux e macOS exigem uma etapa adicional para concluir a instalação.

1. Abrir a janela de linha de comando
2. 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
```

3. 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
   ```

Estabelecimento de uma conexão serial

Para estabelecer uma conexão serial entre sua máquina host e o ESP32 - DevKit C, você deve instalar os drivers CP21 0x 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, você deve instalar o driver da porta COM virtual FTDI. Você pode fazer download desse driver no FTDI.

Para obter mais informações, consulte Estabelecer conexão serial com ESP32. Depois de estabelecer uma conexão serial, anote a porta serial da conexão de sua placa. Você precisará dela durante a compilação da demonstração.

Faça download e configure os FreeRTOS

Depois de configurar seu ambiente, você pode baixar os FreeRTOS em. GitHub Para obter instruções, consulte o arquivo README.md no site. GitHub

Configuração das aplicações de demonstração do FreeRTOS

1. Se você estiver executando macOS ou Linux, abra um prompt de terminal. Se você estiver executando o Windows, abra mingw32.exe. (MinGW é um ambiente de desenvolvimento minimalista para aplicativos nativos do Microsoft Windows.)
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 do AWS Command Line Interface (AWS CLI) para executar AWS IoT comandos. Se você estiver executando o Windows, use o easy_install awscli para instalar o AWS CLI no ambiente mingw32.

   Se você estiver executando o macOS ou o Linux, consulte Installing the AWS CLI.

4. Executar

   ```
   aws configure
   ```

   e configure-o AWS CLI com seu ID de chave de AWS acesso, chave de acesso secreta e AWS região padrão. Para obter mais informações, consulte Configurar a AWS CLI.

5. Use o comando a seguir para instalar o AWS SDK para Python (boto3):

   - No Windows, no ambiente mingw32, 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 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`(WPA2 segurança)

Para executar o script de configuração

1. Se você estiver executando macOS ou Linux, abra um prompt de terminal. Se você estiver executando o Windows, abra o arquivo `mingw32.exe`.
2. 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 da IoT
- Anexa a política da IoT ao certificado e o certificado à coisa da AWS IoT
- Preenche o `aws_clientcredential.h` arquivo com seu AWS IoT endpoint, SSID Wi-Fi e credenciais
- 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`.

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.

Compilar e instalar o 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

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   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
> ```

> **Note**
>
> 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
> ```

Crie e atualize Freertos com CMake

Além do `idf.py` script fornecido pelo SDK do IDF para criar e executar seu código, você também pode criar o projeto com. 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. 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
   ```

Monitoramento de mensagens MQTT na nuvem

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. Navegue até o console do AWS IoT.
2. No painel de navegação, escolha Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*/example/topic** e selecione Inscreva-se no tópico.

Execução das demonstrações de Bluetooth Low-Energy

O FreeRTOS oferece suporte à conectividade 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 Dispositivos Bluetooth Mobile SDKs for FreeRTOS para baixar e instalar o SDK para sua plataforma móvel em seu computador host.
2. Siga as instruções em Aplicativo de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS para configurar o aplicativo móvel de demonstração em seu dispositivo móvel.

Para obter instruções sobre como executar a demonstração do MQTT por Bluetooth Low Energy na sua placa, consulte o Aplicativo de demonstração do MQTT por Bluetooth Low Energy.

Para obter instruções sobre como executar a demonstração do provisionamento de Wi-Fi na sua placa, consulte o aplicativo de demonstração do provisionamento de Wi-Fi.

Usando FreeRTOS em seu próprio projeto para CMake ESP32

Se quiser consumir Freertos em seu CMake próprio projeto, você pode configurá-lo como um subdiretório e construí-lo junto com seu aplicativo. 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.

ESP-IDF v3.3

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

add_executable(my_app src/main.c)

# Tell IDF build to link against this target.
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::mqtt)
```

ESP-IDF v4.2

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Pull in ESP-IDF defined CMake build infrastructure logic to configure the project,
 discover all the components, etc
set(esp_idf_dir "${CMAKE_CURRENT_LIST_DIR}/freertos/vendors/espressif/esp-idf")
include(${esp_idf_dir}/tools/cmake/idf.cmake)

# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)
get_filename_component(
    IDF_EXECUTABLE_SRCS
    "src/main.c" ABSOLUTE
    )
```

Para criar o projeto, execute os CMake comandos a seguir. Verifique se o ESP32 compilador 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
```

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 CMake, você pode encontrar todos os componentes disponíveis na saída resumida. Isso deve ser parecido com a seguinte exemplo.

```
====================Configuration for FreeRTOS====================
  Version:                 201910.00
  Git version:             201910.00-388-gcb3612cb7

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:        ble, ble_hal, ble_wifi_provisioning, common, crypto,
 defender,
                           dev_mode_key_provisioning, freertos_plus_tcp, greengrass,
                           https, kernel, mqtt, ota, pkcs11, pkcs11_implementation,
                           platform, secure_sockets, serializer, shadow, tls, wifi
```

```
  Enabled by user:          ble, ble_hal, ble_wifi_provisioning, defender, greengrass,
                            https, mqtt, ota, pkcs11, pkcs11_implementation, platform,
                            secure_sockets, shadow, wifi
  Enabled by dependency:    common, crypto, demo_base, dev_mode_key_provisioning,
                            freertos, freertos_plus_tcp, kernel, pkcs11_mbedtls,
                            secure_sockets_freertos_plus_tcp, serializer, tls, utils
  3rdparty dependencies:    http_parser, jsmn, mbedtls, pkcs11, tinycbor
  Available demos:          demo_ble, demo_ble_numeric_comparison, demo_defender,
                            demo_greengrass_connectivity, demo_https, demo_mqtt,
 demo_ota,
                            demo_shadow, demo_tcp, demo_wifi_provisioning
  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::mqtt`, `AFR::ota` e assim por diante.

Adicionar componentes personalizados ao ESP-IDF

Você pode adicionar mais componentes ao ambiente de compilação 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.

ESP-IDF v3.3

```
# include paths of this components.
set(COMPONENT_ADD_INCLUDEDIRS include)
```

```
# source files of this components.
set(COMPONENT_SRCDIRS src)
# Alternatively, use COMPONENT_SRCS to specify source files explicitly
# set(COMPONENT_SRCS src/example_component.c)

# add this components, this will define a CMake library target.
register_component()
```

ESP-IDF v4.2

```
# Use idf_component_register to add this component, this will define a CMake library
 target.
# Specify component sources in SRCS.
# Specify component include directories in INCLUDE_DIRS.
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include")
# Standard CMake function can be used to specify dependencies. ${COMPONENT_TARGET}
 is defined
# from esp-idf when you call register_component, by default it's
 idf_component_<folder_name>.
target_link_libraries(${COMPONENT_TARGET} PRIVATE AFR::mqtt)
```

Você também pode especificar dependências usando a CMake função `target_link_libraries` padrão. Observe que o nome de destino do seu componente é armazenado na variável `COMPONENT_TARGET`, definida pelo ESP-IDF.

ESP-IDF v3.3

```
# add this component, this will define a CMake library target.
register_component()

# standard CMake function can be used to specify dependencies. ${COMPONENT_TARGET}
 is defined
# from esp-idf when you call register_component, by default it's
 idf_component_<folder_name>.
target_link_libraries(${COMPONENT_TARGET} PRIVATE AFR::mqtt)
```

ESP-IDF v4.2

```
# add this component, this will define a CMake library target.
```

```
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include")

# standard CMake function can be used to specify dependencies. ${COMPONENT_TARGET}
 is defined
# from esp-idf when you call register_component, by default it's
 idf_component_<folder_name>.
target_link_libraries(${COMPONENT_TARGET} PRIVATE AFR::mqtt)
```

Para componentes ESP, isso é feito definindo duas variáveis, `COMPONENT_REQUIRES` e `COMPONENT_PRIV_REQUIRES`. Consulte Build System (CMake) no Guia de Programação ESP-IDF v4.2.

ESP-IDF v3.3

```
# If the dependencies are from ESP-IDF, use these 2 variables. Note these need to be
# set before calling register_component().
set(COMPONENT_REQUIRES log)
set(COMPONENT_PRIV_REQUIRES lwip)
```

ESP-IDF v4.2

```
# If the dependencies are from ESP-IDF, use these 2 variables (REQUIRES,
 PRIV_REQUIRES)
#   in the idf_component_register call: (Refer here for information on the
#   idf_component_register function: https://docs.espressif.com/projects/esp-idf/en/
v4.2/esp32/api-guides/build-system.html#cmake-component-register)
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include"
                       REQUIRES log
                       PRIV_REQUIRES lwip)
```

Em seguida, no arquivo `CMakeLists.txt` de nível superior, você diz ao ESP-IDF onde encontrar esses componentes. Adicione as seguintes linhas em qualquer lugar antes de `add_subdirectory(freertos)`.

ESP-IDF v3.3

```
# 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})
```

ESP-IDF v4.2

```
# 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
)
idf_build_component(${EXTRA_COMPONENT_DIRS})
```

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, CMake procurará os *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ *freertos*/demos/include/ diretórios e. 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

Caso queira fornecer a sua própria`sdkconfig.default`, você pode definir a CMake variável`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`.

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.

ESP-IDF v3.3

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

add_executable(my_app 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::mqtt)
```

ESP-IDF v4.2

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Pull in ESP-IDF defined CMake build infrastructure logic to configure the project,
 discover all the components, etc
set(esp_idf_dir "${CMAKE_CURRENT_LIST_DIR}/freertos/vendors/espressif/esp-idf")
include(${esp_idf_dir}/tools/cmake/idf.cmake)
# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)
get_filename_component(
    IDF_EXECUTABLE_SRCS
    "src/main.c" ABSOLUTE
    )
# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is an variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/foo" ABSOLUTE
)
idf_build_component(${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::mqtt)
```

Para obter mais informações sobre o sistema de compilação do Espressif com CMake, consulte Sistema de compilação nos guias da API do Espressif

Solução de problemas

- Se você estiver executando o macOS e o sistema operacional não o reconhecer ESP-WROVER-KIT, verifique se você não tem os drivers D2XX instalados. Para desinstalá-los, siga as instruções em FTDI Drivers Installation Guide for macOS X.
- O utilitário de monitor fornecido pelo ESP-IDF (e chamado usando make monitor) 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 a comunicação com o gdb sem a necessidade de nenhum hardware JTAG especial. Para obter mais informações, consulte Lançamento do GDB com GDBStub no site do 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. Você pode usar o comutador `-j4` no comando make para reduzir o tempo de compilação:

```
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 Usando FreeRTOS em seu próprio projeto para CMake 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 `CMakeLists.txt` arquivo (no diretório raiz) usando a CMake função padrão`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 vir 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 (ESP-IDF v3.3) ESP-WROVER-KIT

Esta seção mostra como depurar o hardware da Espressif usando o ESP-IDF v3.3. Para depurar usando o ESP-IDF v4.2, consulte Código de depuração no Espressif ESP32 - DevKit C e (ESP-IDF v4.2) ESP-WROVER-KIT.

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, essas são as conexões com o DevKitC. ESP32

Cor do fio C232HM-DDHSL-0	ESP32 Pino GPIO	Nome do sinal JTAG
Marrom (pino 5)	IO14	TMS
Amarelo (pino 3)	IO12	TDI
Preto (pino 10)	GND	GND
Laranja (pino 2)	IO13	TCK
Verde (pino 4)	IO15	TDO

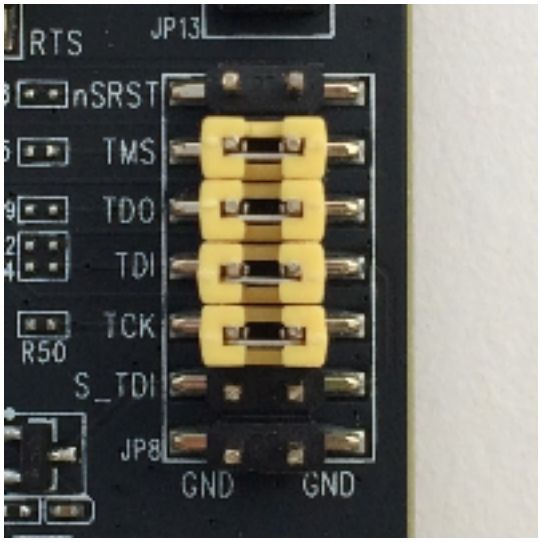
ESP-WROVER-KIT Configuração do JTAG

Para o cabo FTDI C232HM-DDHSL-0, essas são as conexões com o -WROVER-KIT. ESP32

Cor do fio C232HM-DDHSL-0	ESP32 Pino GPIO	Nome do sinal JTAG
Marrom (pino 5)	IO14	TMS
Amarelo (pino 3)	IO12	TDI
Laranja (pino 2)	IO13	TCK
Verde (pino 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 o JTAG no ESP-WROVER-KIT, coloque os jumpers nos pinos TMS, TDO, TDI, TCK e S_TDI, conforme mostrado aqui.



Depuração no Windows (ESP-IDF v3.3)

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 (ESP-IDF v3.3) ESP-WROVER-KIT. 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 Properties (Propriedades).

   > **Note**
   >
   > O dispositivo pode estar listado como USB Serial Port (Porta serial USB).

   Na janela de propriedades, escolha a guia Details (Detalhes) para ver as propriedades do dispositivo. Se o dispositivo não estiver listado, instale o driver do Windows para FTDI C232HM-DDHSL-0.

3. Na guia Detalhes, escolha Propriedade e, em seguida, escolha Hardware IDs. 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 eles IDs correspondem à IDs entrada`projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Eles IDs são especificados em uma linha que começa com, `ftdi_vid_pid` seguida por uma ID do fornecedor e uma 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 lista suspensa no campo de driver de destino, escolha a seta e selecione Instalar driver. Escolha Replace Driver (Substituir driver).

7. Abra um prompt de comando, navegue até `projects/espressif/esp32/make/aws_demos` e execute o seguinte comando.

   Para ESP32 -WROOM-32 e -WROVER ESP32

   ```
   openocd.exe -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
   ```

   Para ESP32 -SOLO-1

   ```
   openocd.exe -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
   ```

Deixe o prompt de comando aberto.

8. Abra um novo prompt de comando, navegue até o diretório `msys32` e execute `mingw32.exe`.
9. No terminal mingw32, navegue até `projects/espressif/esp32/make/aws_demos` e execute

```
make flash monitor
```

10. Abra outro terminal mingw32, navegue até `projects/espressif/esp32/make/aws_demos` e aguarde até que a demonstração comece a ser executada na placa. Quando isso ocorrer, execute `xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf`. O programa deve parar na função `main`.

> **Note**
>
> ESP32 Ele suporta no máximo dois pontos de ruptura.

Depuração no macOS (ESP-IDF v3.3)

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. 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
```

7. 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)
   ```

8. 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 o IDs para corresponder ao IDs da `system_profiler` saída na etapa anterior.

9. Abra uma janela de terminal, navegue até `projects/espressif/esp32/make/aws_demos` e use o seguinte comando para executar OpenOCD:

   Para ESP32 -WROOM-32 e -WROVER: ESP32

   ```
   openocd -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
   ```

   Para ESP32 -SOLO-1:

   ```
   openocd -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
   ```

10. Abra um novo terminal e use o seguinte comando para carregar o driver de porta serial FTDI.

    ```
    sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
    ```

11. Navegue até `projects/espressif/esp32/make/aws_demos` e execute o seguinte comando.

    ```
    make flash monitor
    ```

12. Abra um novo terminal, navegue até `projects/espressif/esp32/make/aws_demos` e execute o seguinte comando.

```
xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf
```

O programa deve parar em `main()`.

Depuração no Linux (ESP-IDF v3.3)

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 (UART) ESP32 do e é usada para carregar o código na memória flash ESP32 do.

5. Em uma janela de terminal, navegue até `projects/espressif/esp32/make/aws_demos` e use o seguinte comando para executar OpenOCD:

Para ESP32 -WROOM-32 e -WROVER: ESP32

```
openocd -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
```

Para ESP32 -SOLO-1:

```
openocd -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
```

6. Abra outro terminal, navegue até `projects/espressif/esp32/make/aws_demos` e execute o seguinte comando.

```
make flash monitor
```

7. Abra outro terminal, navegue até `projects/espressif/esp32/make/aws_demos` e execute o seguinte comando:

```
xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf
```

O programa deve parar em `main()`.

Código de depuração no Espressif ESP32 - DevKit C e (ESP-IDF v4.2) ESP-WROVER-KIT

Esta seção mostra como depurar o hardware da Espressif usando o ESP-IDF v4.2. Para depurar usando o ESP-IDF v3.3, consulte Código de depuração no Espressif ESP32 - DevKit C e (ESP-IDF v3.3) ESP-WROVER-KIT.

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, essas são as conexões com o DevKitC. ESP32

Cor do fio C232HM-DDHSL-0	ESP32 Pino GPIO	Nome do sinal JTAG
Marrom (pino 5)	IO14	TMS
Amarelo (pino 3)	IO12	TDI

Cor do fio C232HM-DDHSL-0	ESP32 Pino GPIO	Nome do sinal JTAG
Preto (pino 10)	GND	GND
Laranja (pino 2)	IO13	TCK
Verde (pino 4)	IO15	TDO

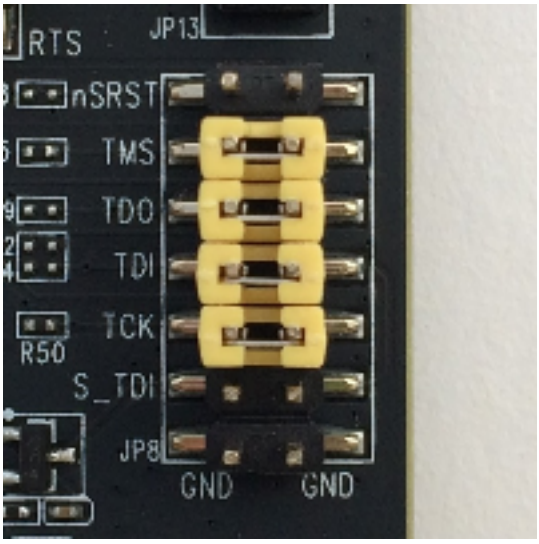
ESP-WROVER-KIT Configuração do JTAG

Para o cabo FTDI C232HM-DDHSL-0, essas são as conexões com o -WROVER-KIT. ESP32

Cor do fio C232HM-DDHSL-0	ESP32 Pino GPIO	Nome do sinal JTAG
Marrom (pino 5)	IO14	TMS
Amarelo (pino 3)	IO12	TDI
Laranja (pino 2)	IO13	TCK
Verde (pino 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 o JTAG no ESP-WROVER-KIT, coloque os 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 (ESP-IDF v4.2) ESP-WROVER-KIT. 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 Properties (Propriedades).

   > **Note**
   >
   > O dispositivo pode estar listado como USB Serial Port (Porta serial USB).

   Na janela de propriedades, escolha a guia Details (Detalhes) para ver as propriedades do dispositivo. Se o dispositivo não estiver listado, instale o driver do Windows para FTDI C232HM-DDHSL-0.
3. Na guia Detalhes, escolha Propriedade e, em seguida, escolha Hardware IDs. 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 eles IDs correspondem à IDs entrada`projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Eles IDs são especificados em uma linha que começa com, `ftdi_vid_pid` seguida por uma ID do fornecedor e uma 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 lista suspensa 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**
>
> ESP32 Ele suporta no máximo dois pontos de ruptura.

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. 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
   ```

7. 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)
   ```

8. 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 o IDs para corresponder ao IDs da `system_profiler` saída na etapa anterior.

9. 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.

10. Abra um novo terminal e use o seguinte comando para carregar o driver de porta serial FTDI.

```
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

11. Navegue até a raiz do diretório de downloads do FreeRTOS e execute

```
idf.py flash monitor
```

12. 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 (UART) ESP32 do e é usada para carregar o código na memória flash ESP32 do.

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()`.

Introdução ao ESP32 Espressif -WROOM-32SE

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Siga este tutorial para começar a usar o ESP32 Espressif -WROOM-32SE. Para comprar um de nosso parceiro no catálogo de dispositivos AWS parceiros, consulte ESP32-WROOM-32SE.

> **Note**
>
> Atualmente, a porta FreeRTOS ESP32 para -WROOM-32SE não suporta o recurso de multiprocessamento simétrico (SMP).

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 aplicativos incorporados para a placa do microcontrolador
3. Fazer a compilação cruzada de um aplicativo de demonstração do FreeRTOS para uma imagem binária
4. Carregar a imagem binária do aplicativo na placa e executar o aplicativo
5. Monitorar e depurar o aplicativo 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 a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma ligação ou mensagem de texto 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 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, você pode visualizar a atividade atual da sua conta e gerenciar sua conta acessando https://aws.amazon.com/e escolhendo 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 a 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. Habilita o Centro de Identidade do IAM.

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

2. No Centro de Identidade do IAM, conceda o acesso administrativo a 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 o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

  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 Criar um conjunto de permissões 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 Adicionar grupos 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.

> **Note**
>
> O FreeRTOS versão 202012.00 oferece suporte ao ESP-IDF v3.3. Se você estiver usando uma revisão posterior do FreeRTOS, siga as instruções específicas IDFv4 de .2 em. Conceitos básicos do ESP-IDF v4.2

Conceitos básicos do ESP-IDF v3.3

Esta seção mostra como usar o ESP-IDF v3.3 no hardware da Espressif. Para usar o ESP-IDF v4.2, consulte Conceitos básicos do ESP-IDF v4.2.

Configuração do hardware Espressif

Consulte o Guia de introdução ESP32 - DevKit C para obter informações sobre como configurar o hardware da placa de desenvolvimento ESP32 -WROOM-32SE.

> **Important**
>
> Quando chegar à seção Get Started do guia do Espressif, siga as etapas abaixo.

Configuração do ambiente de desenvolvimento

Para se comunicar com sua placa, você deve baixar e instalar uma cadeia de ferramentas.

Configuração da cadeia de ferramentas

> **Note**
>
> O ESP-IDF v3.3 (a versão que o FreeRTOS usa) não suporta a versão mais recente do compilador. ESP32 Use o compilador que é compatível com a versão 3.3 do ESP-IDF. Consulte os links anteriores. Para verificar a versão do compilador, execute o comando a seguir.
>
> ```
> xtensa-esp32-elf-gcc --version
> ```

Para configurar a cadeia de ferramentas, siga as instruções do sistema operacional de sua máquina host:

- Configuração padrão da cadeia de ferramentas (ESP-IDF v3.3) para Windows
- Configuração padrão da cadeia de ferramentas (ESP-IDF v3.3) para macOS
- Configuração padrão da cadeia de ferramentas (ESP-IDF v3.3) para Linux

> **Important**
>
> Quando você chegar nas instruções de "Obter ESP-IDF" em Next Steps (Próximas etapas), pare e retorne para as instruções desta página.
> Se você seguiu as instruções de “Get ESP-IDF” em Next Steps agora ou em uma ocasião anterior, você deve limpar a variável de ambiente `IDF_PATH` do sistema antes de continuar.
> Se você seguiu as instruções de “Get ESP-IDF”, essa variável de ambiente é definida automaticamente.

Instalar CMake

O sistema de CMake compilação é necessário para criar os aplicativos de demonstração e teste do FreeRTOS para esse dispositivo. O FreeRTOS oferece suporte às versões 3.13 ou posteriores.

Você pode baixar a versão mais recente CMake do em CMake.org. As distribuições binárias e de origem estão disponíveis.

Para obter mais detalhes sobre o uso CMake com FreeRTOS, consulte. Usando CMake com FreeRTOS

Estabelecimento de uma conexão serial

1. Para estabelecer uma conexão serial entre sua máquina host e o ESP32 -WROOM-32SE, instale os drivers CP21 0x USB para UART Bridge VCP. Você pode fazer download desses drivers no Silicon Labs.
2. Siga as etapas para estabelecer uma conexão serial com ESP32.
3. Depois de estabelecer uma conexão serial, anote a porta serial da conexão de sua placa. Você precisará dela durante a compilação da demonstração.

Faça download e configure os FreeRTOS

Depois de configurar seu ambiente, você pode baixar os FreeRTOS em. GitHub Para obter instruções, consulte o arquivo README.md no site. GitHub

> **Important**
>
> O dispositivo ATECC6 08A tem uma inicialização única que é bloqueada no dispositivo na primeira vez em 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. Pule a última etapa Para formatar as credenciais da AWS IoT e siga as etapas abaixo.
2. A Microchip forneceu várias ferramentas de script para ajudar na configuração das peças ATECC6 08A. Navegue até o diretório *`freertos`*`/vendors/microchip/secure_elements/app/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:

    a. Crie e registre uma autoridade de certificação com AWS

b. Gere suas chaves no ATECC6 08A e exporte a chave pública e o número de série do dispositivo

c. 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.

Compilar, instalar e executar o projeto de demonstração do FreeRTOS

Você pode usar CMake para gerar os arquivos de compilação, Make para criar o binário do aplicativo e o utilitário IDF do Espressif para atualizar sua placa.

Compilar o FreeRTOS no Linux ou macOS (ESP-IDF v3.3)

Se estiver usando o Windows, você poderá pular para Compilar o FreeRTOS no Windows (ESP-IDF v3.3).

Use CMake para gerar os arquivos de compilação e, em seguida, use Make para criar o aplicativo.

Para gerar os arquivos de compilação do aplicativo de demonstração com CMake

1. Navegue até a raiz do diretório de downloads do FreeRTOS.
2. Em uma janela de linha de comando, digite o seguinte comando para gerar os arquivos de compilação.

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32
 -S . -B your-build-directory
```

Observações

Para criar o aplicativo para depuração, adicione o sinalizador `-DCMAKE_BUILD_TYPE=Debug`.
Para gerar os arquivos de compilação do aplicativo de teste, adicione o sinalizador `-DAFR_ENABLE_TESTS=1`.
O código fornecido pelo Espressif usa a pilha de IP leve (lwIP) como a pilha de rede padrão. Para usar a pilha de rede FreeRTOS+TCP em vez disso, adicione o sinalizador ao comando. `–DAFR_ESP_FREERTOS_TCP` CMake

Para adicionar a dependência do LWIP para código não fornecido pelo fornecedor, adicione as linhas a seguir ao arquivo `CMakeLists.txt` de dependência,, para seu CMake componente Wi-Fi personalizado.

```
# Add a dependency on the bluetooth espressif component to the common
 component
set(COMPONENT_REQUIRES lwip)
```

Para criar o aplicativo com Make

1. Navegue até o diretório `build`.
2. Em uma janela de linha de comando, digite o seguinte comando para compilar o aplicativo com Make.

```
make all -j4
```

> **Note**
>
> Você deve gerar os arquivos de compilação com o comando cmake toda vez que alternar entre o projeto `aws_demos` e o projeto `aws_tests`.

Compilar o FreeRTOS no Windows (ESP-IDF v3.3)

No Windows, você deve especificar um gerador de compilação para CMake. Caso contrário, o CMake padrão é Visual Studio. O Espressif recomenda oficialmente o sistema de compilação Ninja, porque ele funciona no Windows, no Linux e no MacOS. Você deve executar CMake comandos em um ambiente nativo do Windows, como cmd ou PowerShell. A execução de CMake comandos em um ambiente Linux virtual, como MSYS2 ou WSL, não é suportada.

Use CMake para gerar os arquivos de compilação e, em seguida, use Make para criar o aplicativo.

Para gerar os arquivos de compilação do aplicativo de demonstração com CMake

1. Navegue até a raiz do diretório de downloads do FreeRTOS.
2. Em uma janela de linha de comando, digite o seguinte comando para gerar os arquivos de compilação.

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32
 -GNinja -S . -B your-build-directory
```

> **Note**
>
> Para criar o aplicativo para depuração, adicione o sinalizador `-DCMAKE_BUILD_TYPE=Debug`.
> Para gerar os arquivos de compilação do aplicativo de teste, adicione o sinalizador `-DAFR_ENABLE_TESTS=1`.
> O código fornecido pelo Espressif usa a pilha de IP leve (lwIP) como a pilha de rede padrão. Para usar a pilha de rede FreeRTOS+TCP em vez disso, adicione o sinalizador ao comando. `–DAFR_ESP_FREERTOS_TCP` CMake
> Para adicionar a dependência do LWIP para código não fornecido pelo fornecedor, adicione as linhas a seguir ao arquivo `CMakeLists.txt` de dependência,, para seu CMake componente Wi-Fi personalizado.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

Para o build do aplicativo

1. Navegue até o diretório `build`.
2. Em uma janela de linha de comando, insira o seguinte comando para invocar o Ninja para compilar o aplicativo.

   ```
   ninja
   ```

   Ou use a CMake interface genérica para criar o aplicativo.

   ```
   cmake --build your-build-directory
   ```

> **Note**
>
> Você deve gerar os arquivos de compilação com o comando cmake toda vez que alternar entre o projeto `aws_demos` e o projeto `aws_tests`.

Instalar e executar o FreeRTOS (ESP-IDF v3.3)

Use o utilitário IDF da Espressif (*`freertos`*`/vendors/espressif/esp-idf/tools/idf.py`) para atualizar sua placa, executar o aplicativo e visualizar os logs.

Para apagar o flash da placa, navegue até o diretório *`freertos`* e insira o seguinte comando.

```
./vendors/espressif/esp-idf/tools/idf.py erase_flash -B build-directory
```

Para atualizar o binário do aplicativo para a placa, use `make`:

```
make flash
```

Você também pode usar o script do IDF para atualizar a placa.

```
./vendors/espressif/esp-idf/tools/idf.py flash -B build-directory
```

Use os comandos a seguir para visualizar os logs:

```
./vendors/espressif/esp-idf/tools/idf.py monitor -p /dev/ttyUSB1 -B build-directory
```

> **Tip**
>
> Você também pode combinar esses comandos.
>
> ```
> ./vendors/espressif/esp-idf/tools/idf.py erase_flash flash monitor -p /dev/
> ttyUSB1 -B build-directory
> ```

Conceitos básicos do ESP-IDF v4.2

Esta seção mostra como usar o ESP-IDF v4.2 no hardware da Espressif. Para usar o ESP-IDF v3.3, consulte Conceitos básicos do ESP-IDF v3.3.

> **Note**
>
> Os comandos do Linux neste tutorial exigem o uso do shell Bash.

Configuração do hardware Espressif

Para obter informações sobre como configurar o hardware da placa de desenvolvimento ESP32 - WROOM-32SE, consulte o Guia de introdução ESP32- DevKit C V4.

> **Important**
>
> Quando chegar à seção Get Started dos guias do Espressif, siga as etapas abaixo.

Configuração do ambiente de desenvolvimento

Para se comunicar com a sua placa, você precisa fazer download e instalar uma cadeia de ferramentas.

Configuração da cadeia de ferramentas

Para configurar a cadeia de ferramentas, siga as instruções do sistema operacional de sua máquina host:

- Configuração padrão do conjunto de ferramentas e pré-requisitos (ESP-IDF v4.2) para Windows
- Configuração padrão do conjunto de ferramentas e pré-requisitos (ESP-IDF v4.2) para macOS
- Configuração padrão do conjunto de ferramentas e pré-requisitos (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.

Concluir a instalação (Linux/Mac)

O instalador do ESP-IDF para Windows instala todas as ferramentas necessárias. As plataformas Linux e MAC OSX exigem uma etapa adicional para concluir a instalação.

Abra a janela de linha de comando, 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
```

Em seguida, adicione as ferramentas da cadeia de ferramentas ESP-IDF ao caminho do terminal com o comando a seguir.

```
source vendors/espressif/esp-idf/export.sh
```

Estabelecimento de uma conexão serial

1. Para estabelecer uma conexão serial entre sua máquina host e o ESP32 -WROOM-32SE, instale os drivers CP21 0x USB para UART Bridge VCP. Você pode fazer download desses drivers no Silicon Labs.
2. Siga as etapas para estabelecer uma conexão serial com ESP32.
3. Depois de estabelecer uma conexão serial, anote a porta serial da conexão de sua placa. Você precisará dela durante a compilação da demonstração.

Faça download e configure os FreeRTOS

Depois de configurar seu ambiente, você pode baixar os FreeRTOS em. GitHub Para obter instruções, consulte o arquivo README.md no site. GitHub

> **Important**
>
> O dispositivo ATECC6 08A tem uma inicialização única que é bloqueada no dispositivo na primeira vez em 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. Ao chegar à última etapa Para formatar as credenciais da AWS IoT , execute as etapas a seguir.
2. A Microchip forneceu várias ferramentas de script para ajudar na configuração das peças ATECC6 08A. Navegue até o diretório *freertos*`/vendors/microchip/secure_elements/app/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:

   a. Crie e registre uma autoridade de certificação com AWS

   b. Gere suas chaves no ATECC6 08A e exporte a chave pública e o número de série do dispositivo

   c. Gerar um certificado para o dispositivo e registrar esse certificado na AWS
4. Carregar o certificado CA e o certificado de dispositivo no dispositivo seguindo as instruções para Provisionamento de chaves no modo de desenvolvedor.

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.

Compilar e instalar o 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

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   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
> ```

> **Note**
>
> 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
> ```

Crie e atualize Freertos com CMake

Além de usar o script idf.py fornecido pelo SDK do IDF para criar e executar seu código, você também pode criar o projeto com. 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
```

Monitorando mensagens MQTT na nuvem AWS

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*/example/topic** e selecione Inscreva-se no tópico.

Mais informações

Para obter informações adicionais sobre como usar e solucionar problemas com ESP32 placas Espressif, consulte os tópicos a seguir:

- Usando FreeRTOS em seu próprio projeto para CMake ESP32
- Solução de problemas
- Depuração

Introdução ao kit de conectividade IoT Infineon XMC48 00

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 USB/Serial converter, in addition to the XMC4800 IoT Connectivity Kit.

The CP2104 is a common USB/Serial conversor de 3,3 V que esteja amplamente disponível em placas como a 04 Friend da Adafruit. CP21

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. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilar uma aplicação 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 ambiente de desenvolvimento

O FreeRTOS usa o ambiente de desenvolvimento DAVE da Infineon para programar o 00. XMC48 Antes de começar, você precisa fazer download e instalar 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. Depois de registrar na Infineon, 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 a sonda de depuração integrada da placa XMC48 00 Relax EtherCAT, você precisa dos drivers incluídos no pacote de software e documentação da 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 serial UART nos pinos P0.0 e P0.1, que são rotulados na serigrafia da placa 00 XMC48 Relax EtherCAT. Para configurar uma conexão serial:

1. Conecte o pino chamado "RX<P0.0" ao pino "TX" do conversor USB/serial.
2. Conecte o pino chamado "TX<P0.1" ao pino "RX" do conversor USB/serial.
3. Conecte o pino de aterramento do conversor serial a um dos pinos "GND" na 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 XMC48 00 e o módulo Wi-Fi Click requerem 3,3V. 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.

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), Import (Importar). Na janela Import (Importar), expanda a pasta Infineon, escolha DAVE Project (Projeto do DAVE) e, em seguida, escolha Next (Próximo).
3. Na janela Importar projetos DAVE, escolha Selecionar diretório raiz, escolha Procurar e, em seguida, escolha o projeto de demonstração XMC48 00.

   No diretório em que você descompactou o download do FreeRTOS, 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.

Execução do projeto de demonstração do FreeRTOS

1. Use um cabo USB para conectar seu kit de conectividade XMC48 00 IoT ao seu computador. A placa tem dois conectores microUSB. 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 em `aws_demos`, escolha Debug As (Depurar como) e DAVE C/C++ Application (aplicação DAVE C/C++).
4. Clique duas vezes em GDB SEGGER J-Link Debugging (Depuração de GDB SEGGER J-Link) 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 cliente MQTT das etapas 4-5 deve exibir as mensagens MQTT enviadas pelo seu dispositivo. Se você usar a conexão serial, verá algo semelhante ao seguinte 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----
```

Crie a demonstração do FreeRTOS com CMake

Se você preferir não usar um IDE para o desenvolvimento de Freertos, você pode CMake usar como alternativa para criar e executar os aplicativos de demonstração ou os 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, consulteUsando CMake com FreeRTOS. (MinGW é um ambiente de desenvolvimento minimalista para aplicações nativas do Microsoft Windows.)

Para criar a demonstração do FreeRTOS com CMake

1. Configure o conjunto de ferramentas GNU Arm Embedded.

   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 CMake Pré-requisitos.

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_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 comando a seguir para criar 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 flash:

```
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 aplicação devem estar visíveis por meio da conexão de série estabelecida com a placa.

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*/example/topic** e selecione Inscreva-se no tópico.

Solução de problemas

Se ainda não o fez, certifique-se de configurar AWS IoT e baixar seus FreeRTOS para conectar seu dispositivo à nuvem. AWS Para obter instruções, consulte Primeiras etapas.

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 conectividade XMC48 IoT Infineon OPTIGA Trust X e 00

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Infineon OPTIGA Trust X Secure Element e o kit de conectividade 00 XMC48 IoT. 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 - Infineon 00 XMC48 IoT Connectivity Kit, visite o Catálogo de dispositivos do AWS parceiro para comprar um de nosso parceiro.
2. Pacote de extensões de segurança:
   - Secure Element - Infineon OPTIGA Trust X.

     Visite o Catálogo de dispositivos do AWS parceiro para comprá-los do nosso parceiro.
   - Placa de personalização – Placa de personalização OPTIGA da Infineon.
   - Placa adaptadora - Adaptador Infineon MyIo T.

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, você precisa de um conversor USB/serial de 3,3 V, além do kit de conectividade 00 XMC48 IoT. Sabe-se que o conversor USB/serial JBtek EL-PN-47310126 funciona nesta demonstração. Você também precisa de três male-to-male fios de ligação (para receber (RX), transmitir (TX) e aterrar (GND)) para conectar o cabo serial à placa adaptadora Infineon MyIo T.

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 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 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.
4. Para fins de monitoramento e depuração, interagir com a aplicação 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 00. XMC48 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 a sonda de depuração integrada do kit de conectividade XMC48 00 IoT, você precisa dos drivers incluídos no pacote de software e documentação da 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 XMC48 00 e o módulo Wi-Fi Click requerem 3,3V. 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

Você pode usar o cliente MQTT no AWS IoT console para monitorar as mensagens que seu dispositivo envia para a AWS nuvem. Convém configurar isso antes que o dispositivo execute o projeto de demonstração.

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 Test (Teste) 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

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 Importar projetos DAVE, escolha Selecionar diretório raiz, escolha Procurar e, em seguida, escolha o projeto de demonstração XMC48 00.

   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 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.

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 (aplicação 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, acesse o AWS IoT console.

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----
```

Crie a demonstração do FreeRTOS com CMake

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, consulteUsando CMake com FreeRTOS. (MinGW é um ambiente de desenvolvimento minimalista para aplicações nativas do Microsoft Windows.)

Se você preferir não usar um IDE para o desenvolvimento de Freertos, você pode CMake usá-lo 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 demonstração do FreeRTOS com CMake

1. Configure o conjunto de ferramentas GNU Arm Embedded.

   a. Faça o 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 CMake e MinGW.

   Para obter instruções, consulte CMake Pré-requisitos.

3. Crie uma pasta para conter os arquivos de construção 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 construção (*build-folder*) e use o comando a seguir para criar 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 flash:

```
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 da aplicação 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, acesse o AWS IoT console. 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 MW32x AWS IoT Starter Kit

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O AWS IoT Starter Kit é um kit de desenvolvimento baseado no 88 MW32 0/88MW322, 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 MW322 módulos 88 MW32 0/88 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
- Configuração do ambiente de desenvolvimento
- Compilação e execução do projeto de demonstração do FreeRTOS
- Depuração
- Solução de problemas

Configuração do hardware

Conecte a MW32x placa ao seu 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 UARTx porta física 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.

```
marvell@pe-lt586:amzsdk$
marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -
Bbuild -DAFR_ENABLE_TESTS=0

====================Configuration for Amazon FreeRTOS====================
  Version:                 v1.2.4
  Git version:             AMZSDK_V1.2.r6.p1-12-gdd17d10

Target microcontroller:
  vendor:                  Marvell
  board:                   mw300_rd
  description:             Marvell Board for AmazonFreeRTOS
  family:                  Wireless Microcontroller
  data ram size:           512KB
  program memory size:     2MB

Host platform:
  OS:                      Linux-4.15.0-47-generic
  Toolchain:               arm-gcc
  Toolchain path:          /home/marvell/Software/gcc-arm-none-eabi-4_9-2015q3
  CMake generator:         Unix Makefiles

Amazon FreeRTOS modules:
  Modules to build:        kernel, freertos_plus_tcp, bufferpool, crypto, greengrass, mqtt
, ota, pkcs11, secure_sockets, shadow, tls, wifi
  Disabled by user:
  Disabled by dependency:  posix

  Available demos:         demo_key_provisioning, demo_logging, demo_mqtt_hello_world, dem
o_mqtt_pubsub, demo_tcp, demo_shadow, demo_greengrass, demo_ota
  Available tests:         test_crypto, test_greengrass, test_mqtt, test_ota, test_pkcs11,
 test_secure_sockets, test_tls, test_shadow, test_wifi, test_memory
=========================================================================

-- Configuring done
-- Generating done
-- Build files have been written to: /home/marvell/gerrit/amzsdk/build
marvell@pe-lt586:amzsdk$
```

2. 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:

```
[ 92%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder_close
_container_checked.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborerrorstrings.
c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser_dup_st
ring.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborpretty.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/jsmn/jsmn.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/logging/aws_logging_task_dyna
mic_buffers.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/demo_runner/aws_demo_runner.c
.obj
[ 95%] Linking C static library afr_ota.a
[ 95%] Built target afr_ota
[ 96%] Linking C executable aws_demos.axf
[100%] Built target aws_demos
marvell@pe-lt586:build$
```

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----
```

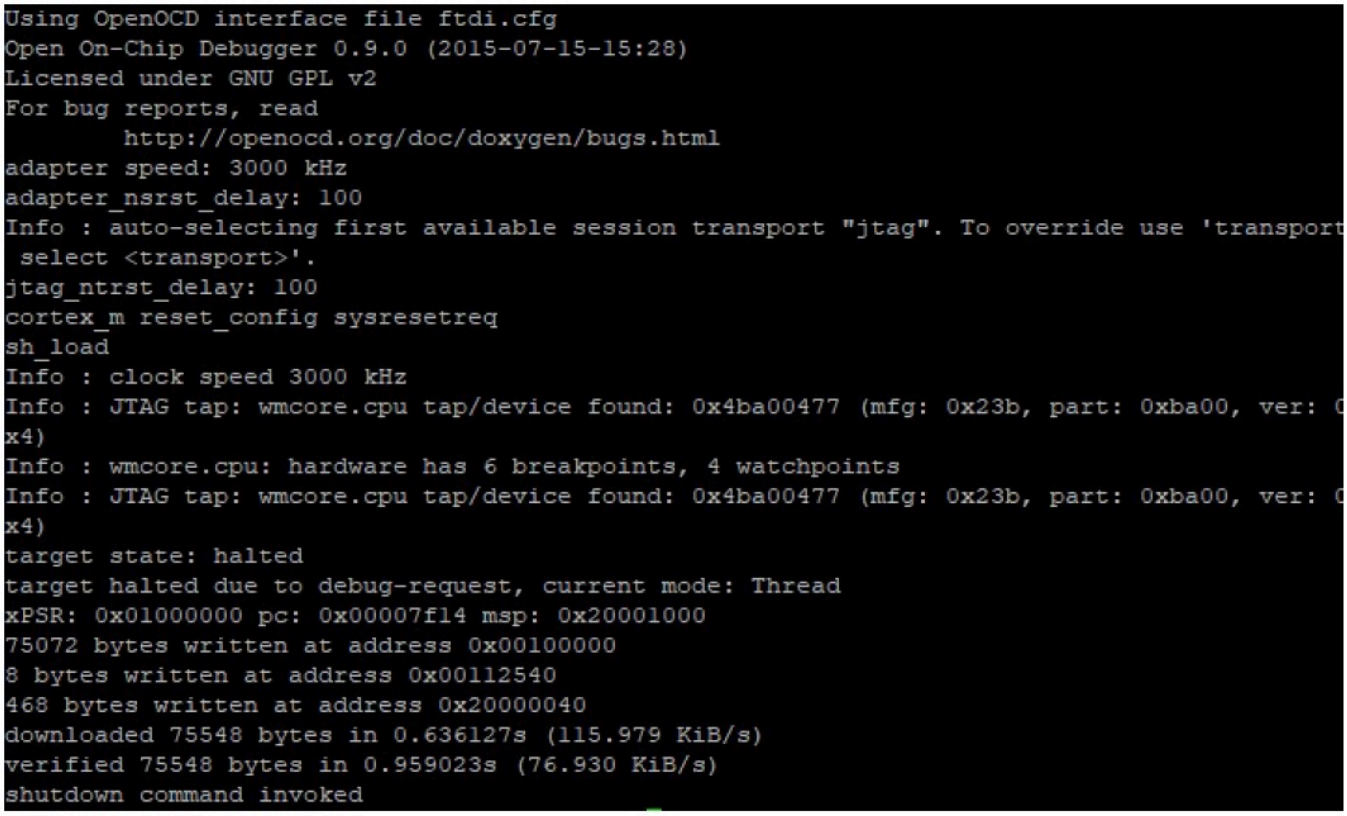
7. (Opcional) Como método alternativo para testar a imagem, use o utilitário flashprog para copiar a imagem do microcontrolador do host diretamente na RAM do microcontrolador. A imagem não é copiada na instalação, então ela será perdida após a reinicialização do microcontrolador.

   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`.

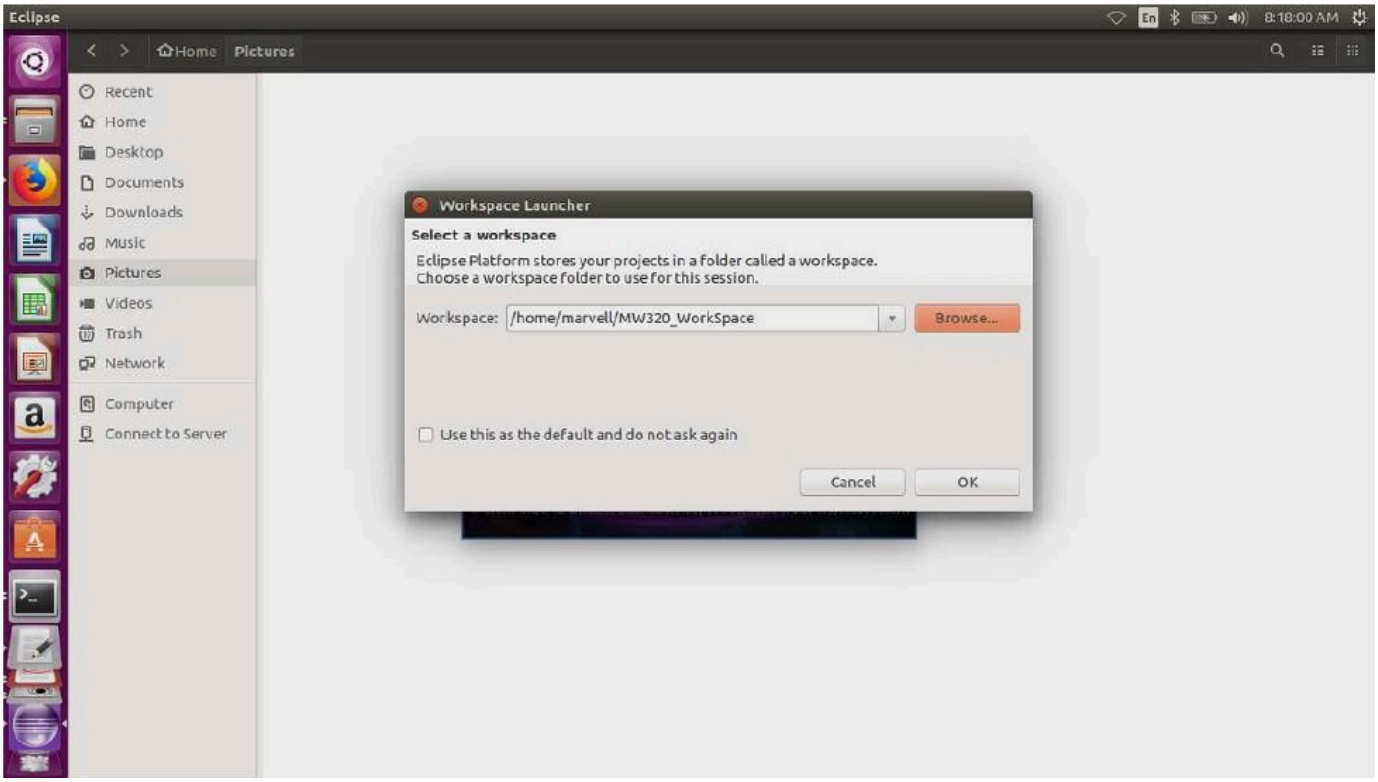
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Execute o comando a seguir para trabalhar com o projeto Eclipse do `aws_tests`.

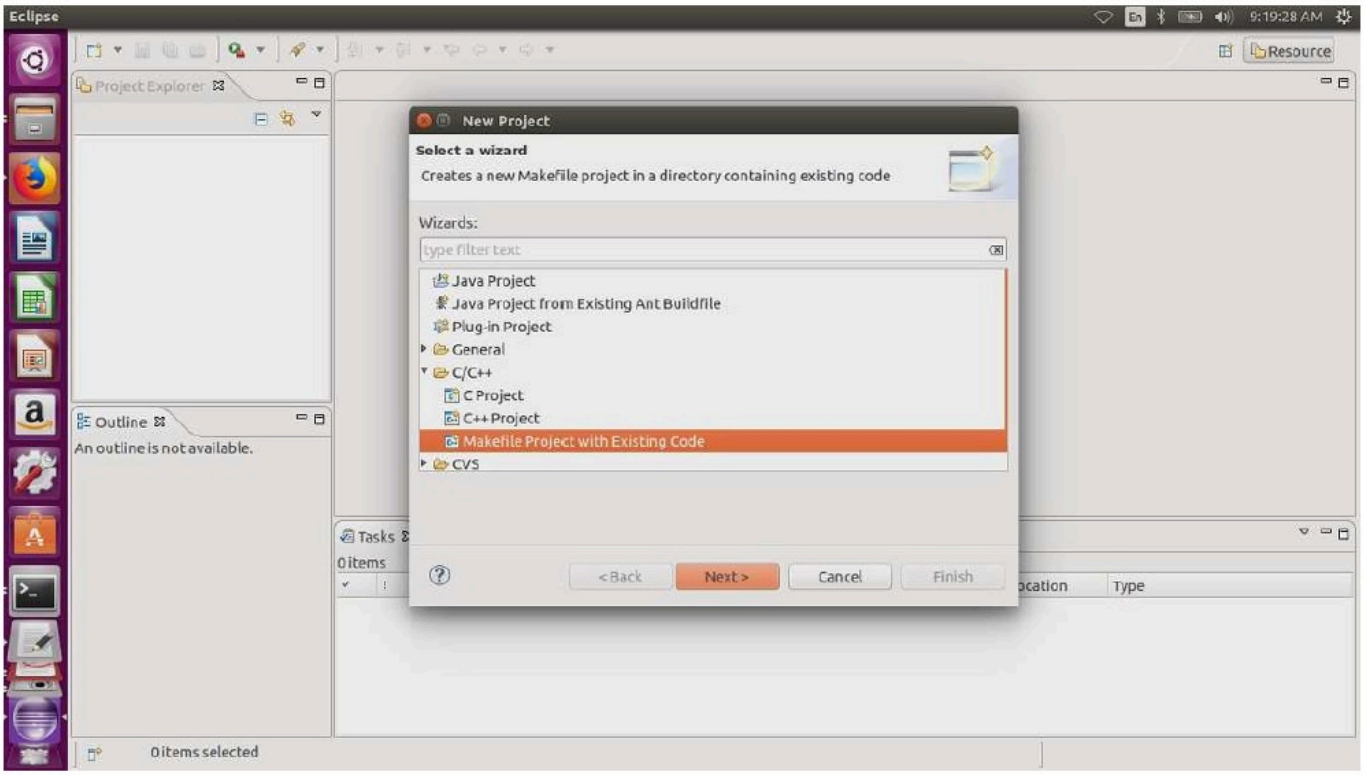
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> Execute o comando `cmake` toda vez que você alternar entre o projeto `aws_demos` e o `aws_tests`.

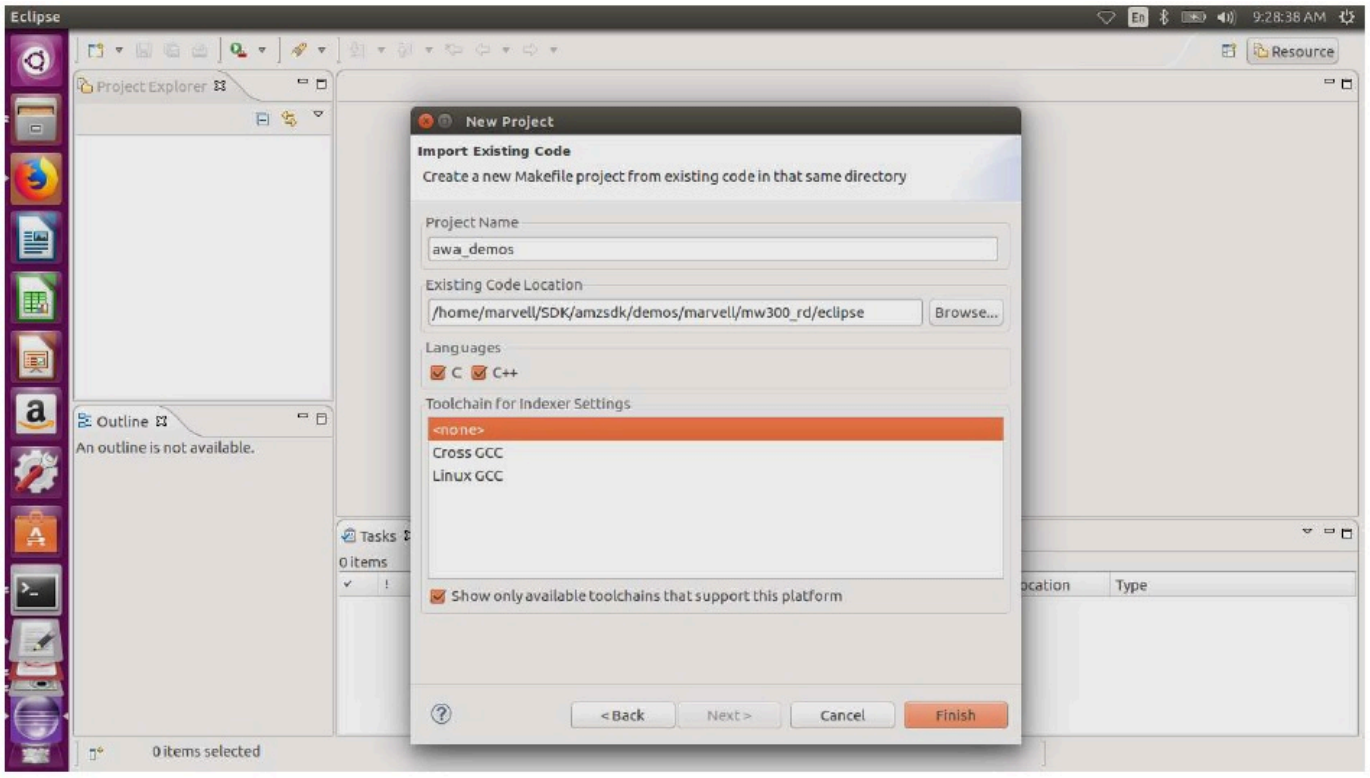
2. Abra o Eclipse e, quando solicitado, escolha a área de trabalho do Eclipse, conforme mostrado na figura abaixo.



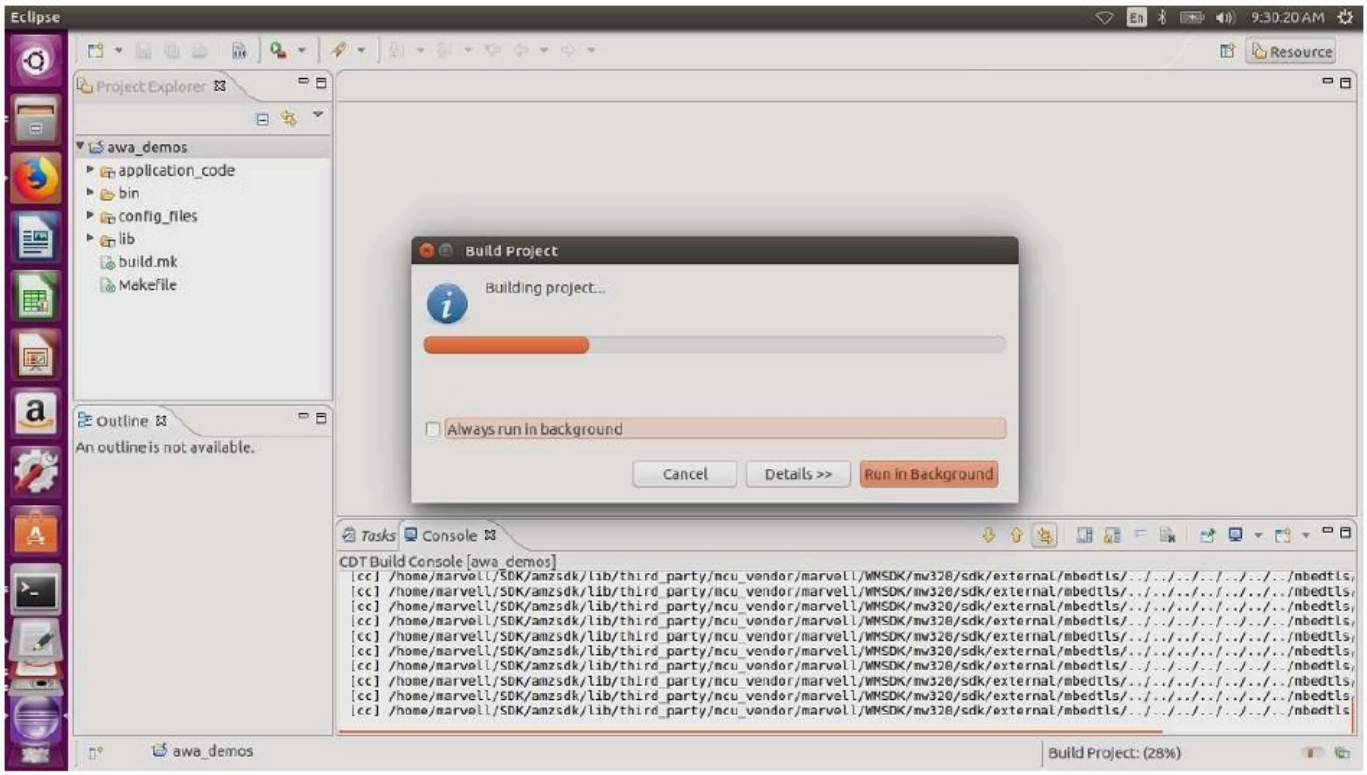
3. Escolha a opção de criar um projeto Makefile: com código existente, conforme mostrado na figura abaixo.



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.

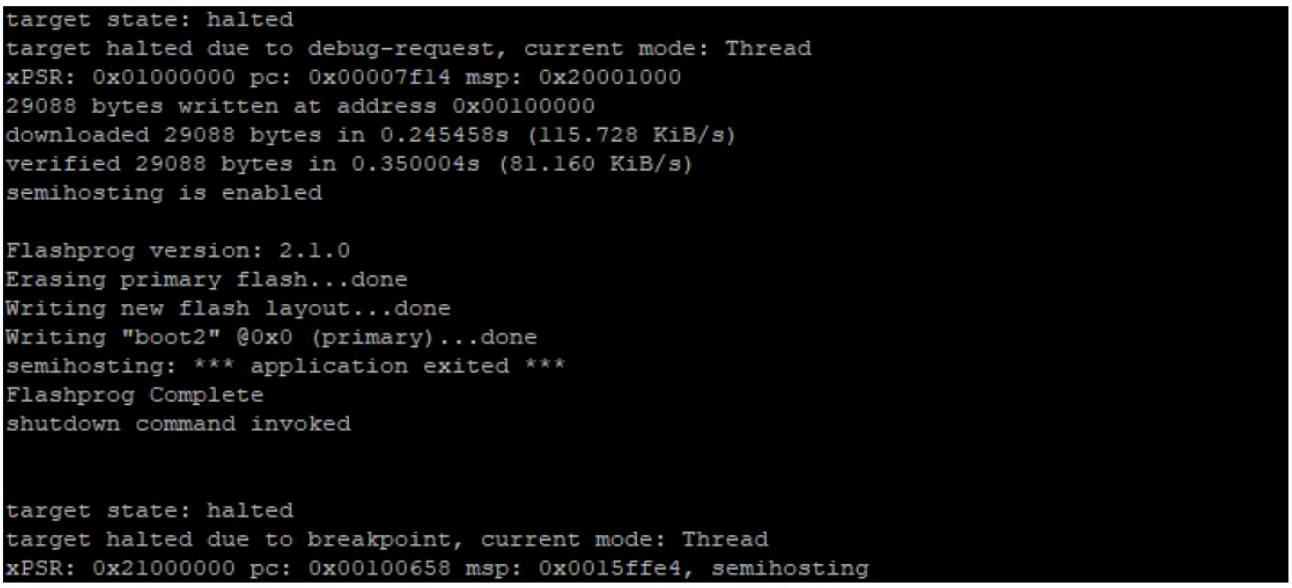
   a. Antes de instalar a imagem do firmware, prepare a instalação da placa de desenvolvimento com os componentes comuns, Layout e Boot2. Use os seguintes comandos.

   ```
   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.

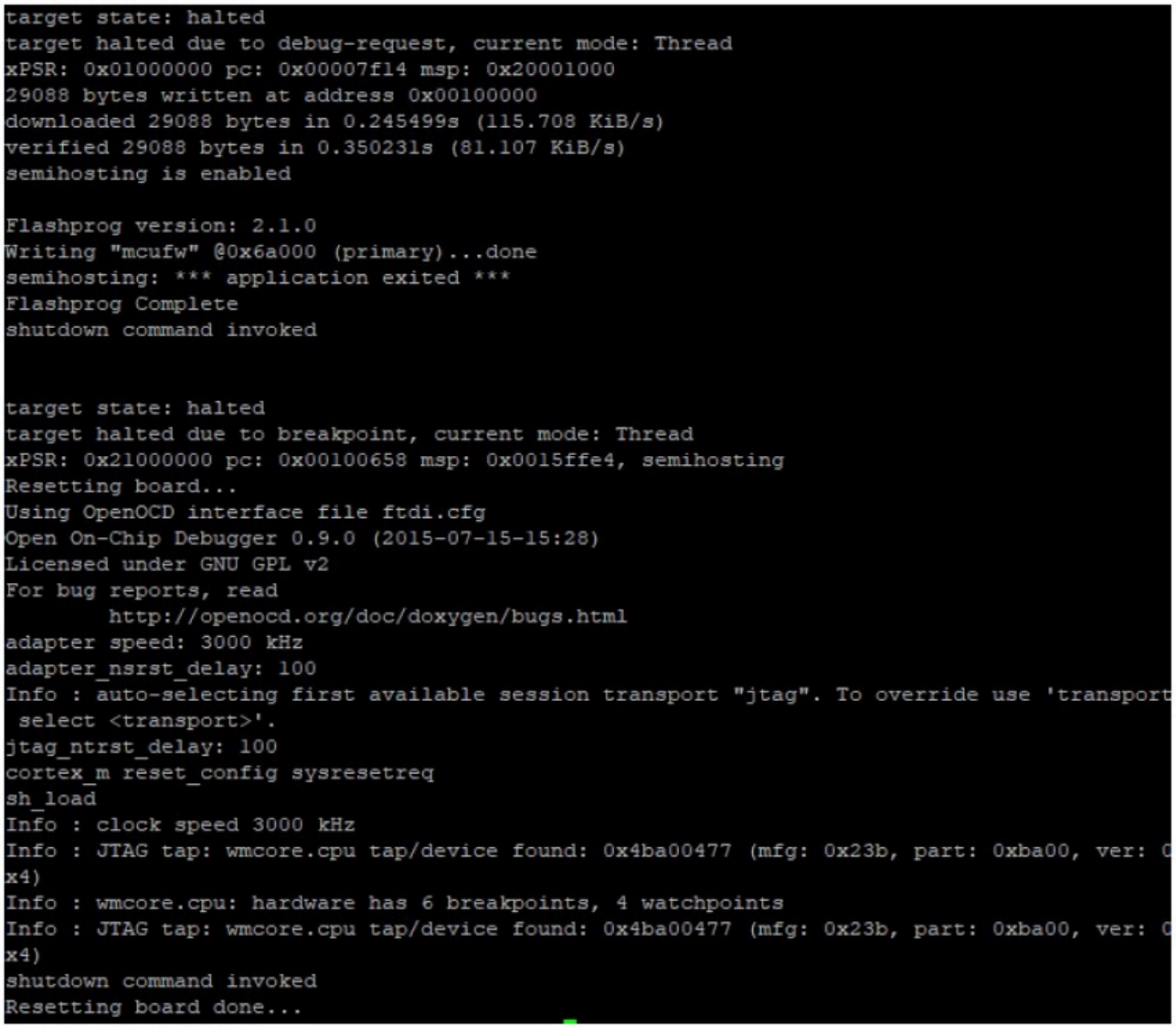
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

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.

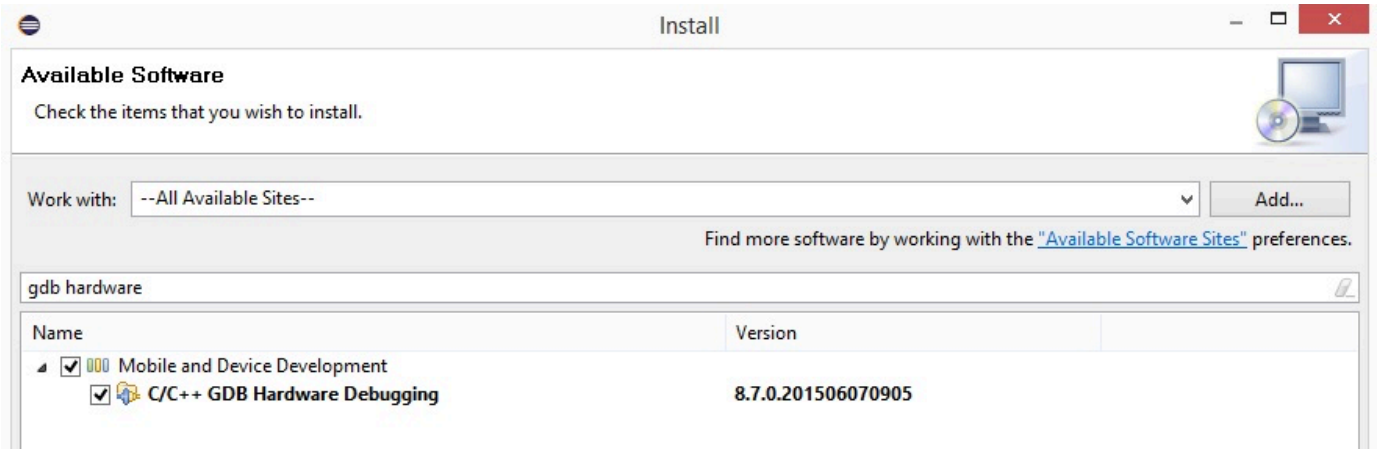
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

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
```

Introdução ao kit de desenvolvimento MediaTek MT7697 Hx

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o MediaTek MT7697 Hx Development Kit. Se você não tiver o kit de desenvolvimento MediaTek MT7697 Hx, visite o Catálogo de dispositivos do AWS 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. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilar uma aplicação 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 ambiente de desenvolvimento

Antes de configurar seu ambiente, conecte seu computador à porta USB do MediaTek MT7697 Hx Development Kit.

Download e instalação do Keil MDK

Você pode usar o Kit de desenvolvimento de microcontrolador (MDK) Keil baseado em GUI para configurar, compilar e executar projetos do FreeRTOS na placa. O Keil MDK inclui o IDE do µVision e o Depurador do µVision.

> **Note**
>
> O Keil MDK é compatível somente com máquinas Windows 7, Windows 8 e Windows 10 de 64 bits.

Para fazer download e instalar o Keil MDK

1. Acesse a página Conceitos básicos do Keil MDK e escolha Download MDK-Core (Fazer download do MDK-Core).
2. Insira e envie suas informações para serem registradas com o Keil.
3. Clique com o botão direito no arquivo executável do MDK e salve o instalador do Keil MDK no seu computador.
4. Abra o instalador do Keil MDK e siga as etapas para a conclusão. Certifique-se de instalar o pacote de MediaTek dispositivos (MT76x7 Série).

Estabelecimento de uma conexão serial

Para estabelecer uma conexão serial com o MediaTek MT7697 Hx Development Kit, você deve instalar o driver da porta serial Arm Mbed Windows. Você pode fazer download do driver em Mbed. Siga as etapas na página do driver serial do Windows para baixar e instalar o driver do MediaTek MT7697 Hx Development Kit.

Depois que você instalar o driver, uma porta COM aparecerá 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

> **Note**
>
> Se você tiver problemas para se conectar à sua conta depois de instalar o driver, poderá ser necessário reinicializar a máquina.

Compilação e execução do projeto de demonstração do FreeRTOS com o MDK Keil

Para compilar o projeto de demonstração do FreeRTOS no 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`.

Como executar o projeto de demonstração do FreeRTOS

1. Defina o MediaTek MT7697 Hx Development Kit para o modo PROGRAM.

   Para definir o kit no modo PROGRAM, mantenha pressionado o botão PROG. Com o botão PROG ainda pressionado, pressione e solte o botão RESET e, depois, solte o botão 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 Use (Usar), defina o depurador como CMSIS-DAP Debugger (Depurador CMSIS-DAP) 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 botão RESET no seu kit de desenvolvimento MediaTek MT7697 Hx.

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***`your-thing-name`*****`example/topic`** e selecione Inscreva-se no tópico.

Solução de problemas

Depuração de projetos do FreeRTOS no Keil µVision

Atualmente, você deve editar o MediaTek pacote incluído no Keil µVision antes de poder depurar o projeto de demonstração do FreeRTOS com Keil µVision. MediaTek

Para editar o MediaTek pacote para depuração de projetos FreeRTOS

1. Localize e abra o arquivo `Keil_v5\ARM\PACK\.Web\MediaTek.MTx.pdsc` na pasta de instalação do Keil MDK.
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 Debug (Depurar) e, depois, CMSIS-DAP Debugger (Depurador CMSIS-DAP).
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. Defina o MediaTek MT7697 Hx Development Kit para o modo PROGRAM.

   Para definir o kit no modo PROGRAM, mantenha pressionado o botão PROG. Com o botão PROG ainda pressionado, pressione e solte o botão RESET e, depois, solte o botão PROG.
7. No menu, escolha Flash (Atualizar) e Download (Fazer download).

   O µVision notificará você quando o download for concluído.
8. Pressione o botão RESET no seu kit de desenvolvimento MediaTek MT7697 Hx.
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 ) ; }
```

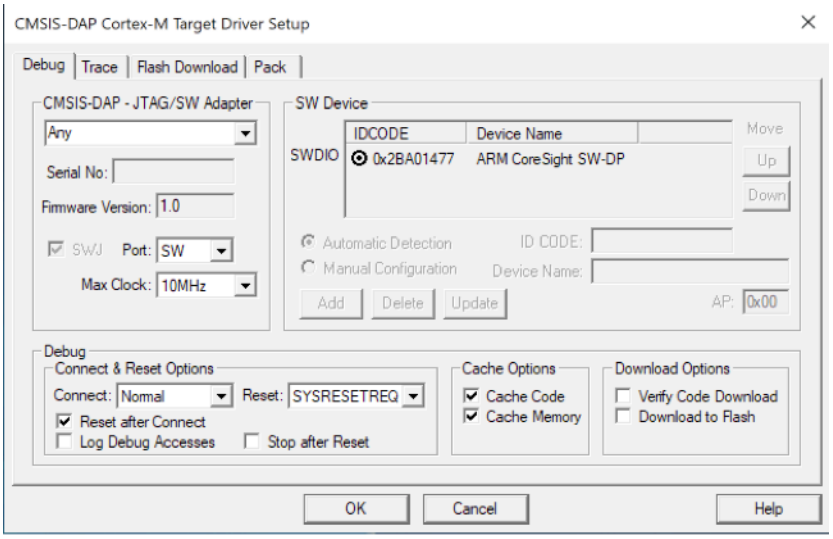
11. Na janela Call Stack + Locals (Chamar pilha + locais), altere o valor de `wait_ice` para `0`.
12. Defina os pontos de interrupção no código-fonte do projeto e execute-o.

Solução de problemas de configuração do depurador IDE

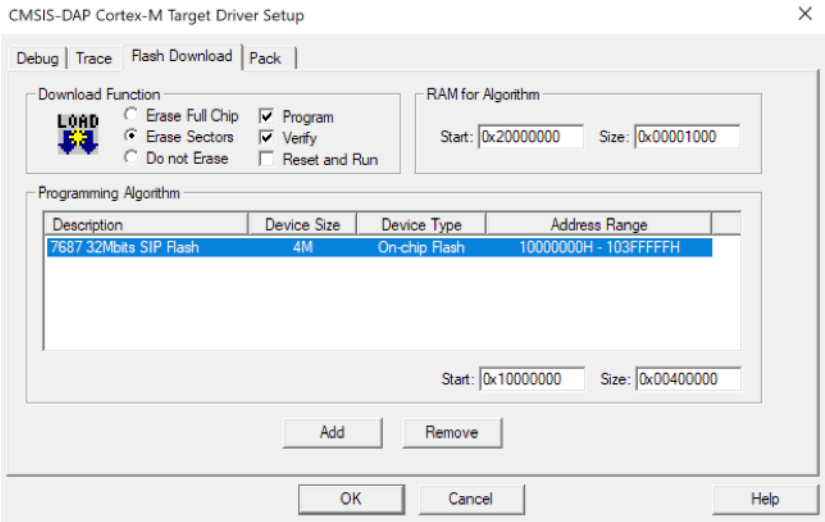
Se você está com problemas para depurar uma aplicação, as configurações do seu depurador podem estar incorretas.

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 mais informações sobre soluções de problemas gerais sobre os Conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Começando com o Microchip Curiosity PIC32 MZ EF

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Microchip Curiosity PIC32 MZ EF. Se você não tiver o pacote Microchip Curiosity PIC32 MZ EF, visite o Catálogo de dispositivos do AWS parceiro para comprar um de nosso parceiro.

O pacote inclui os seguintes itens:

- Placa de desenvolvimento do Curiosity PIC32 MZ EF
- MikroElectronika Placa de clique USB UART
- MikroElectronika WiFi 7 cliques em Board
- PIC32 LAN872Placa filha 0 PHY

Você também precisa os seguintes itens para depuração:

- Depurador MPLAB Snap In-Circuit
- Kit de PICkit 3 cabos de programação (opcional)

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.

> **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.

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.
5. Interagir com o aplicativo em execução na placa em uma conexão serial para fins de monitoramento e depuração.

Configurar o hardware Microchip Curiosity PIC32 MZ EF

1. Conecte a placa MikroElectronika USB UART click ao conector MicroBUS 1 no Microchip PIC32 Curiosity MZ EF.

2. Conecte a placa-filha PIC32 LAN872 0 PHY ao conector J18 no Microchip Curiosity PIC32 MZ EF.
3. Conecte a placa de clique MikroElectronika USB UART ao seu computador usando um cabo USB A para USB mini-B.
4. Para conectar a placa à Internet, use uma das seguintes opções:

   - Para usar o Wi-Fi, conecte a placa de clique MikroElectronika Wi-Fi 7 ao conector MicroBus 2 no Microchip Curiosity PIC32 MZ EF. Consulte Configuração das demonstrações do FreeRTOS.
   - Para usar a Ethernet para conectar a placa Microchip Curiosity PIC32 MZ EF à Internet, conecte a placa-filha PIC32 LAN872 0 PHY ao conector J18 no Microchip Curiosity MZ EF. PIC32 Conecte uma extremidade de um cabo Ethernet à placa-filha LAN872 0 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 conector ICSP no Microchip Curiosity PIC32 MZ EF.
6. Conecte uma extremidade do cabo ICSP do kit de PICkit 3 cabos de programação ao Microchip Curiosity PIC32 MZ EF.

   Se você não tiver o kit de PICkit 3 cabos de programação, você pode usar jumpers de fio M-F Dupont para conectar 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 cabeamento para o Pino 1 no Microchip Curiosity PIC32 MZ EF, representado pelo círculo branco, esteja alinhado com o Pino 1 no MPLAB Snap Debugger.

   Para obter mais informações sobre o Depurador MPLAB Snap, consulte Ficha de informações do depurador no circuito MPLAB Snap.

Configure o hardware Microchip Curiosity PIC32 MZ EF usando o 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 PICkit integrado a bordo (PKOB) seguindo estas etapas.

1. Conecte a placa MikroElectronika USB UART click ao conector MicroBUS 1 no Microchip PIC32 Curiosity MZ EF.
2. Para conectar a placa à Internet, siga um destes procedimentos:
   - Para usar o Wi-Fi, conecte a placa de clique MikroElectronika Wi-Fi 7 ao conector MicroBus 2 no Microchip Curiosity PIC32 MZ EF. (Siga as etapas em “Como configurar seu Wi-Fi” em Configuração das demonstrações do FreeRTOS).
   - Para usar a Ethernet para conectar a placa Microchip Curiosity PIC32 MZ EF à Internet, conecte a placa-filha PIC32 LAN872 0 PHY ao conector J18 no Microchip Curiosity MZ EF. PIC32 Conecte uma extremidade de um cabo Ethernet à placa-filha LAN872 0 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 PIC32 MZ EF ao seu computador usando um cabo USB tipo A para USB micro-B.
4. Conecte a placa de clique MikroElectronika USB UART ao seu 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 MPLAB que sejam compatíveis com o Harmony v2, como a v2.10 do XC32 compilador MPLAB 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, o AWS FreeRTOS Reference Integrations v202007.00 é suportado somente no 3.5. MPLabv5 As versões anteriores das Integrações de Referência do AWS FreeRTOS são suportadas na versão 4.0. MPLabv5

MPLabv53.5 transferências

- MPLAB X Integrated Development Environment para Windows
- MPLAB X Integrated Development Environment para macOS
- MPLAB X Integrated Development Environment para Linux

MPLab Downloads mais recentes (MPLabv5.40)

- MPLAB X Integrated Development Environment para Windows
- MPLAB X Integrated Development Environment para macOS
- MPLAB X Integrated Development Environment para Linux

3. Instale o compilador MPLAB XC32 :

   - Compilador MPLAB XC32 /32++ para Windows
   - Compilador MPLAB XC32 /32++ para macOS
   - Compilador MPLAB XC32 /32++ 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

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 ***`compiler-name`***(Opções globais) para expandi-la e selecione ***`compiler-name`***-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).

Crie a demonstração do FreeRTOS com CMake

Se você preferir não usar um IDE para o desenvolvimento de Freertos, você pode CMake usar como alternativa para criar e executar os aplicativos de demonstração ou os aplicativos que você desenvolveu usando editores de código e ferramentas de depuração de terceiros.

Para criar a demonstração do FreeRTOS com CMake

1. Crie um diretório para conter os 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 Usando CMake com FreeRTOS.

Para usar a Ethernet para se conectar, você deve definir a macro do pré-processador `PIC32_USE_ETHERNET`.

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.

3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

Solução de problemas

Para obter informações sobre a solução de problemas, consulte Solução de problemas de conceitos básicos.

Começando com o Nordic n 0-DK RF5284

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Nordic n RF5284 0-DK. Se você não tiver o Nordic n RF5284 0-DK, visite o Catálogo de dispositivos do AWS parceiro para comprar um de nosso parceiro.

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 a aplicação móvel de demonstração. Se você estiver usando um dispositivo Android, poderá usar o Android Studio para criar a aplicação 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 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.

Configuração do hardware Nordic

Conecte seu computador host à porta USB denominada J2, localizada diretamente acima do suporte da bateria de célula tipo moeda em sua placa Nordic n 0. RF5284

Para obter mais informações sobre como configurar o Nordic n RF5284 0-DK, consulte o Guia do usuário do kit de desenvolvimento n RF5284 0.

Configuração do ambiente de desenvolvimento

Download e instalação do Segger Embedded Studio

O FreeRTOS suporta o Segger Embedded Studio como um ambiente de desenvolvimento para o Nordic n 0-DK. RF5284

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 da aplicação 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 a aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS em um dispositivo móvel.

Como configurar a aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS

1. Siga as instruções em Celular SDKs para dispositivos Bluetooth 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.

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, Terminal Emulator, Connect ***port*** (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.

Fazer download do FreeRTOS

Para baixar o FreeRTOS para o Nordic RF5284 n 0-DK, acesse a página do GitHub 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.

Configuração do projeto

Para executar a demonstração, você precisa configurar seu projeto para trabalhar com ele AWS IoT. Para configurar seu projeto para funcionar AWS IoT, seu dispositivo deve estar registrado como uma AWS IoT coisa. Você deve ter registrado o dispositivo quando Configure o AWS IoT Amazon Cognito para Freertos Bluetooth Low Energy.

Para configurar seu AWS IoT endpoint

1. Faça login no console do AWS IoT.
2. No painel de navegação, selecione Configurações.

   Seu AWS IoT endpoint aparece na caixa de texto 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). Anote o nome da AWS IoT coisa para o seu dispositivo.
4. Com seu AWS IoT endpoint e o nome do AWS IoT item em mãos, abra *freertos*/demos/include/aws_clientcredential.h em seu IDE e especifique valores para as seguintes #define constantes:

   - clientcredentialMQTT_BROKER_ENDPOINT *Your AWS IoT endpoint*
   - clientcredentialIOT_THING_NAME *Your board's AWS IoT thing name*

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_demos_config.h` e defina `CONFIG_MQTT_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 a aplicação 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 específico do fornecedor UUIDs.
   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

     O tamanho da tabela de atributos em bytes. O tamanho deve ser um múltiplo de 4.

   (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. Escolha Activate Your Free License (Ativar sua 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 da aplicação de demonstração do MQTT por Bluetooth Low Energy para concluir a demonstração com a aplicação 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.

Começando com o Nuvoton 487 NuMaker-IoT-M

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar a placa de desenvolvimento Nuvoton NuMaker-IoT-M 487. O microcontrolador da série inclui módulos RJ45 Ethernet e Wi-Fi integrados. Se você não tiver o Nuvoton NuMaker-IoT-M 487, visite o Catálogo de dispositivos do AWS parceiro para comprar um de 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 baixar a edição Keil MDK Nuvoton com um desconto no preço da série Nuvoton Cortex-M4. MCUs O Keil MDK só é compatível com o Windows.

Para instalar a ferramenta de desenvolvimento para o NuMaker-IoT-M 487

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 Test (Teste) 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 sua placa Numaker-IoT-M 487 à sua 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.

Usando CMake com FreeRTOS

Você também pode usar CMake para criar e executar os aplicativos de demonstração do FreeRTOS ou os aplicativos que você desenvolveu usando editores de código e ferramentas de depuração de terceiros.

Verifique se você instalou o sistema de CMake compilação. Siga as instruções em Usando CMake com 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 Test (Teste) 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. Vamos nos referir a essa pasta como *BUILD_FOLDER* o.
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 intermitente, certifique-se de que o interruptor MSG (nº 4 do ICE) esteja ISW1 ligado. Quando você conecta a placa, uma janela (e unidade) será atribuída. (Consulte Solução de problemas.)

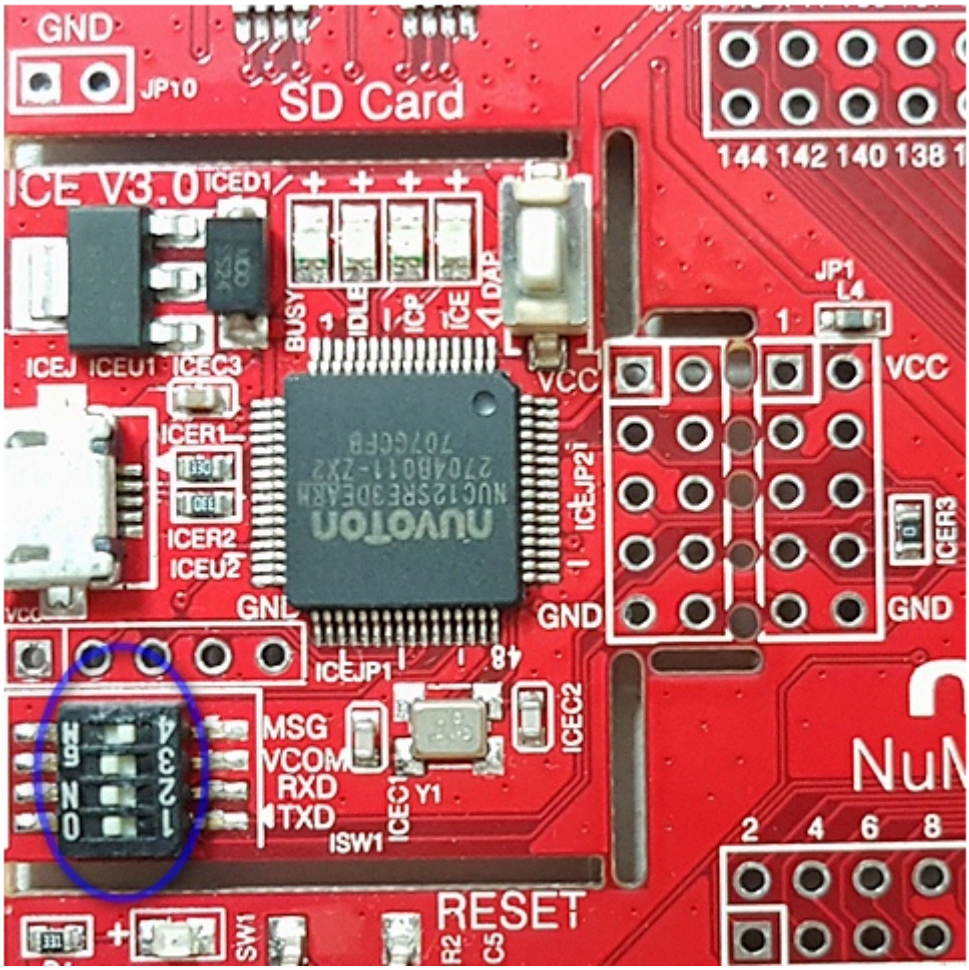
6. Abra um emulador de terminal para visualizar as mensagens via UART. Siga as instruções em Instalação de um emulador de terminal.

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 seu dispositivo ao Keil MDK (IDE) por meio do Nu-Link, certifique-se de que o interruptor MSG (nº 4 do ICE) esteja DESLIGADO, ISW1 conforme mostrado.



Se você tiver problemas para configurar seu ambiente de desenvolvimento ou se conectar à placa, entre em contato com a Nuvoton.

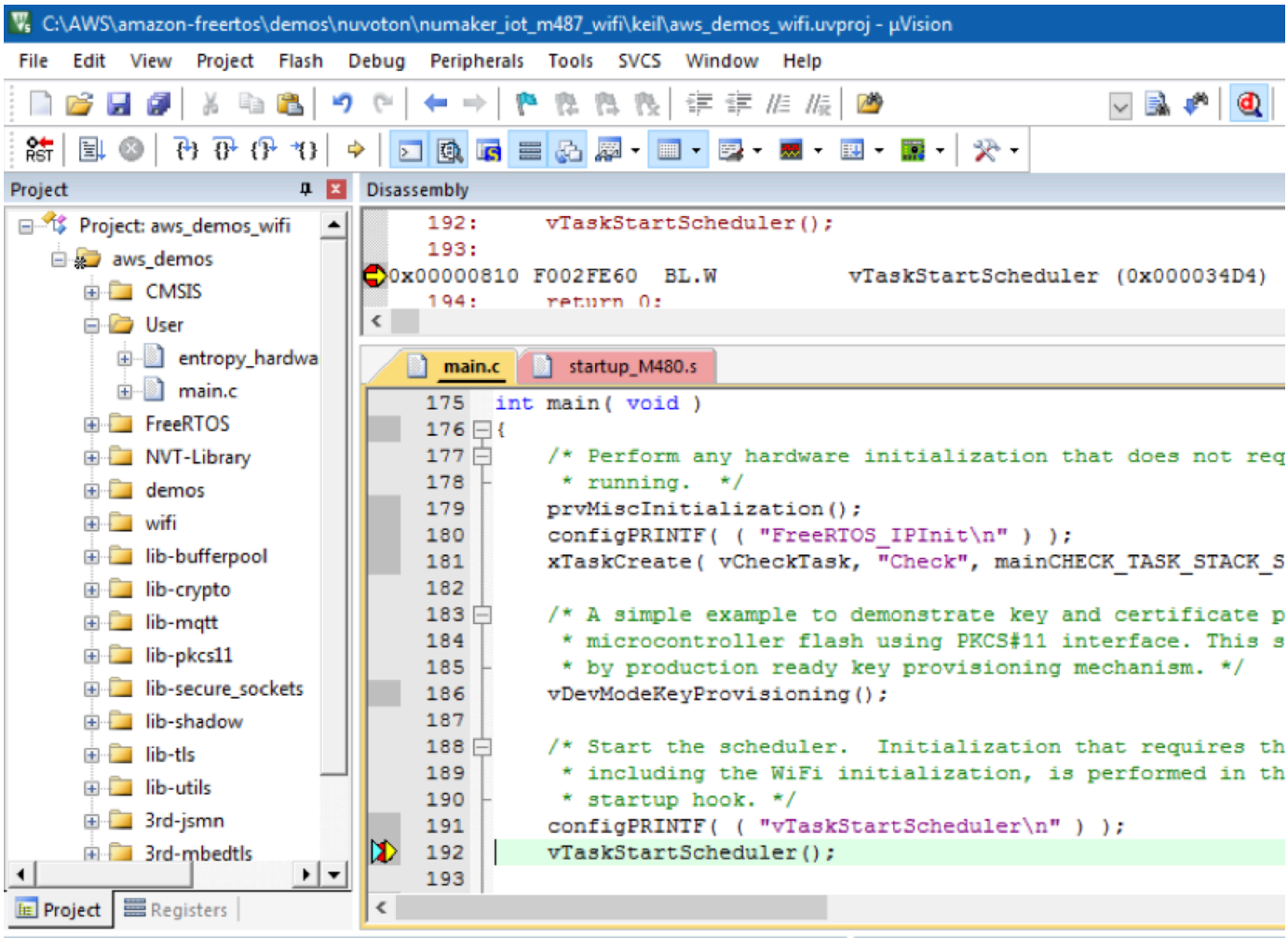
Depuração de projetos do FreeRTOS no Keil µVision

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:

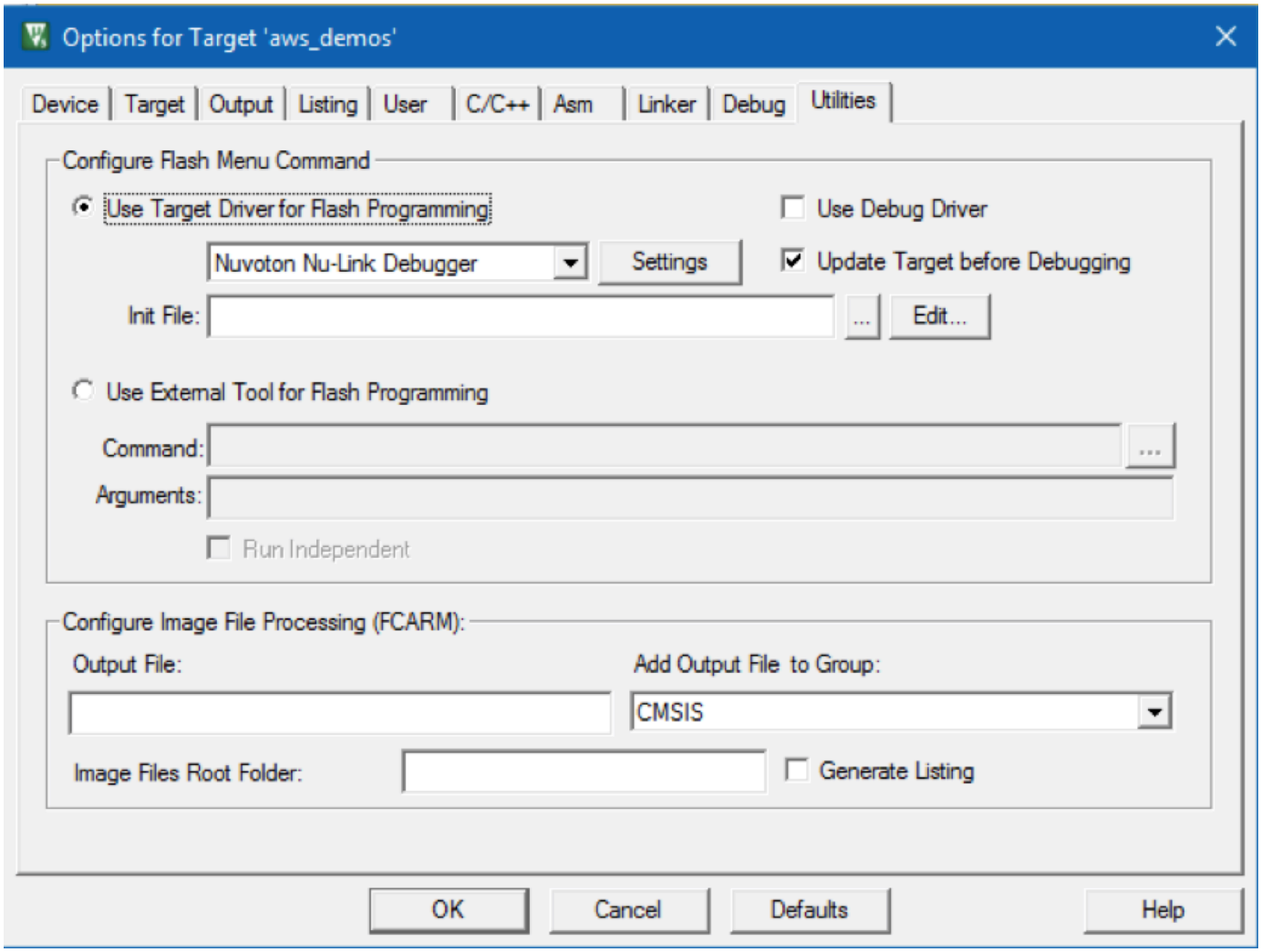


Solução de problemas de configurações de depuração do µVision

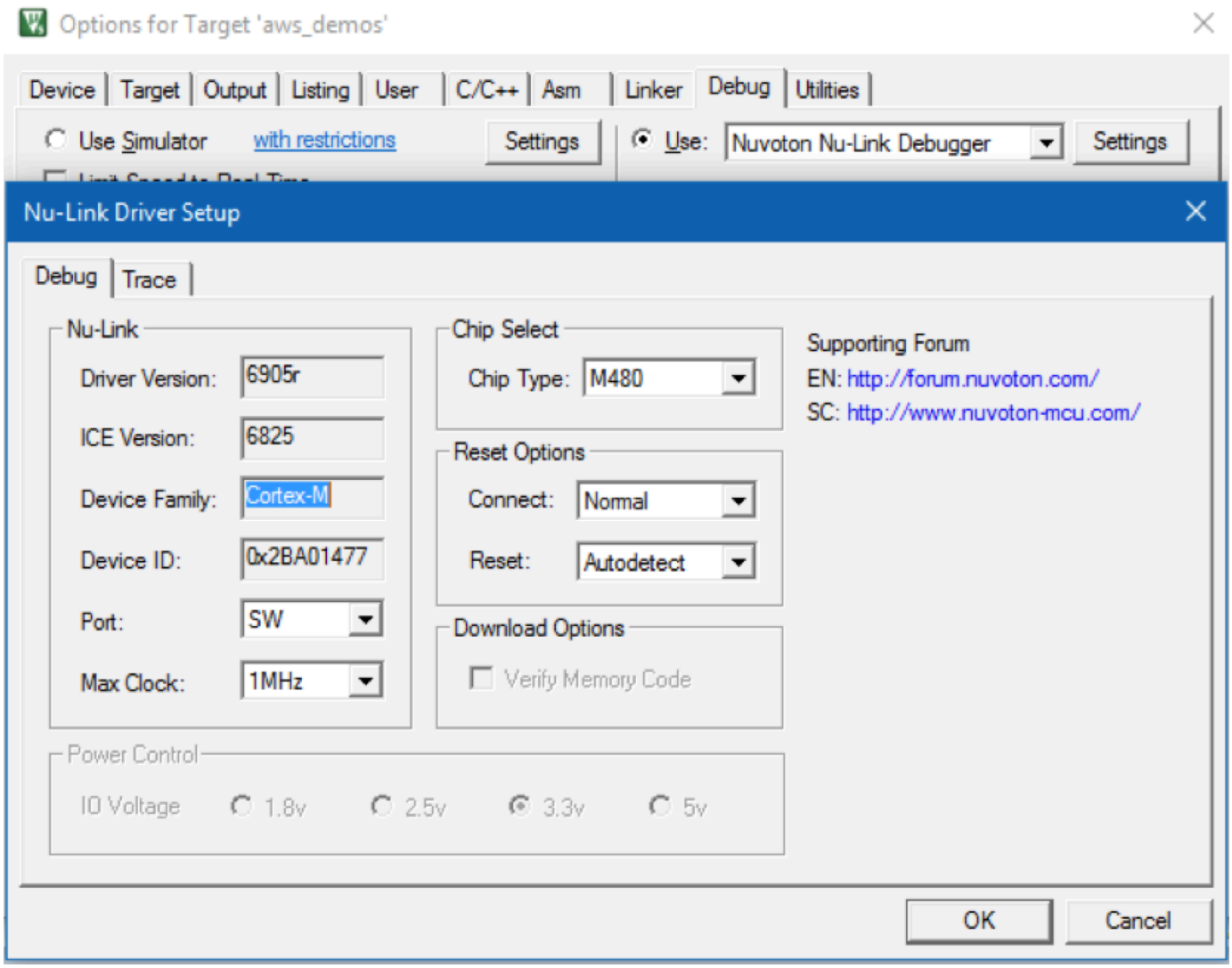
Se você encontrar problemas ao depurar uma aplicação, verifique se as configurações de depuração estão definidas corretamente no Keil µVision.

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 LPC54 IoT NXP 018

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o módulo de LPC54 IoT NXP 018. Se você não tiver um módulo LPC54 IoT NXP 018, visite o Catálogo de dispositivos do AWS parceiro para

comprar um de nosso parceiro. Use um cabo USB para conectar o módulo LPC54 IoT NXP 018 ao computador.

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 NXP

Para configurar o NXP 018 LPC54

- Conecte seu computador à porta USB do NXP LPC54 018.

Para configurar o depurador JTAG

Você precisa de um depurador JTAG para iniciar e depurar seu código em execução na placa NXP 018. LPC54 O FreeRTOS foi testado usando um módulo IoT OM4 0006. Para obter mais informações sobre depuradores compatíveis, consulte o Manual do usuário do módulo IoT NXP LPC54 018, disponível na página do produto do módulo IoT OM4 LPC540007 018.

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 portas USB do seu computador usando cabos mini-USB para USB.

Configuração do ambiente de desenvolvimento

O FreeRTOS suporta IDEs dois para o módulo IoT NXP LPC54 018: IAR Embedded Workbench e. MCUXpresso

Antes de começar, instale um desses IDEs.

Para instalar o IAR Embedded Workbench para ARM

1. Navegue até o IAR Embedded Workbench para ARM e faça download do software.

   > **Note**
   >
   > O IAR Embedded Workbench para ARM requer o Microsoft Windows.

2. Execute o instalador e siga as instruções.
3. No License Wizard (Assistente de licença), escolha Register with IAR Systems to get an evaluation license (Registrar com IAR Systems 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 do NXP

1. Baixe e execute o MCUXpresso instalador do NXP.

   > **Note**
   >
   > As versões 10.3 e posterior são compatíveis.

2. Navegue até o MCUXpresso SDK e escolha Criar seu SDK.

   > **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 Criar MCUXepresso SDK.
7. O SDK para Windows usando o MCUXpresso IDE já foi criado. Escolha Download SDK. Se você estiver usando outro sistema operacional, em Host OS (SO do host), escolha seu sistema operacional e, em seguida, Download SDK (Fazer download do SDK).
8. Inicie o MCUXpresso IDE e escolha a SDKs guia Instalado.
9. Arraste e solte o arquivo do SDK baixado na SDKs janela Instalado.

Se você tiver problemas durante a instalação, consulte Suporte da NPX ou Recursos do desenvolvedor da NPX.

Compilação e execução do projeto de demonstração do FreeRTOS

Importação da demonstração do FreeRTOS para o IDE

Como importar o código de exemplo do FreeRTOS para o IDE do IAR Embedded Workbench

1. Abra o IAR Embedded Workbench e, no menu File (Arquivo), escolha Open Workspace (Abrir Workspace).
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 de amostra do FreeRTOS para o IDE MCUXpresso

1. Abra e MCUXpresso, 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).

Execução do projeto de demonstração do FreeRTOS

Como executar o projeto de demonstração do FreeRTOS com o IDE do IAR Workbench Embedded

1. No IDE, no menu Project (Projeto), escolha Make.
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 demonstração do FreeRTOS com o IDE MCUxpresso

1. No IDE, no menu Project (Projeto), escolha Build (Criar).
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).

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*/example/topic** e selecione Inscreva-se no tópico.

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.

Começando com o Renesas Starter Kit+ para N-2MB RX65

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Renesas Starter Kit+ para N-2MB. RX65 Se você não tiver o Renesas RSK+ para RX65 N-2MB, visite o Catálogo de dispositivos de AWS parceiros e compre um de nossos 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

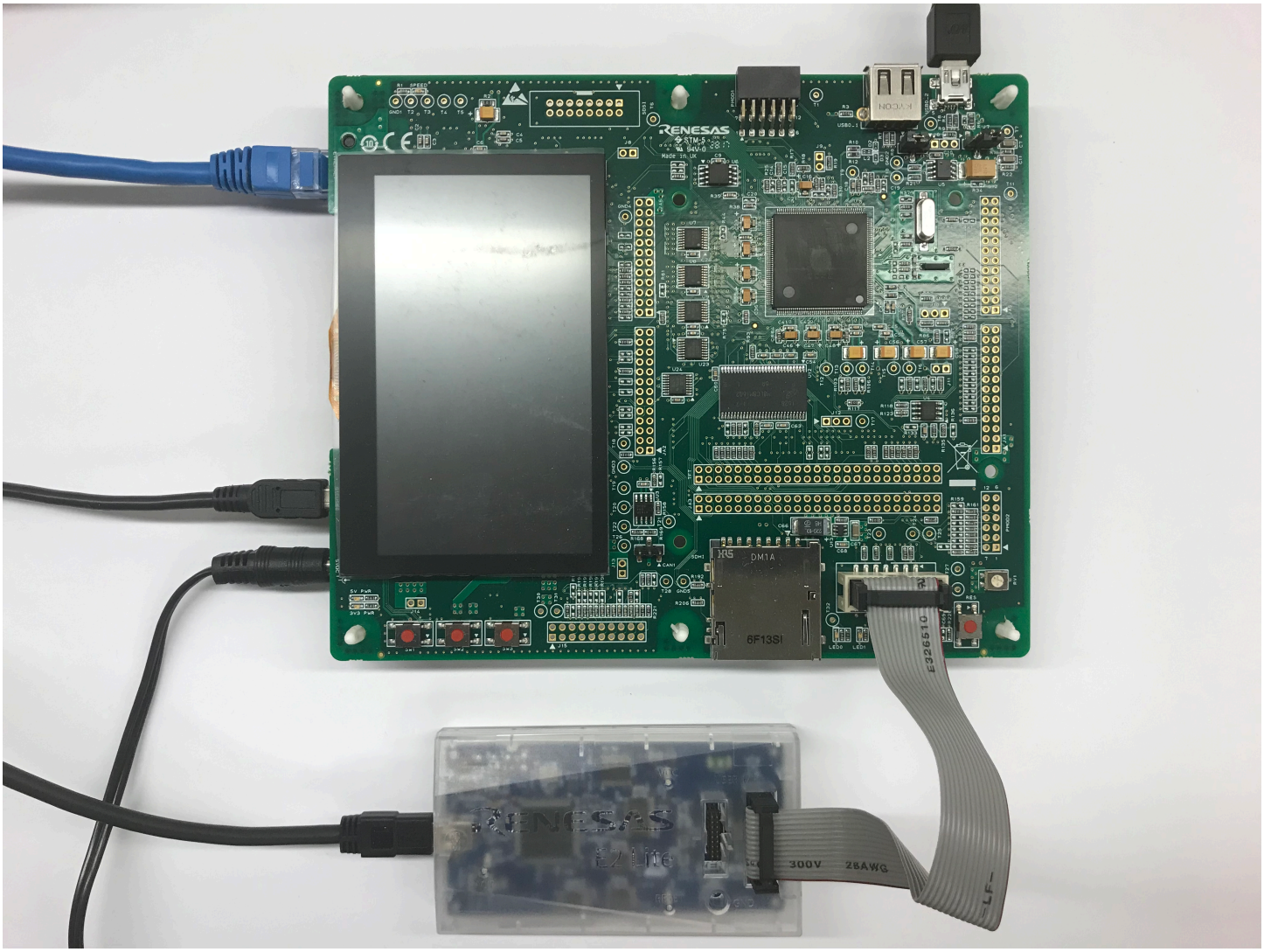
Para configurar o RSK+ para N-2MB RX65

1. Conecte o adaptador de alimentação positivo de +5V ao conector PWR no RSK+ para N-2MB. RX65
2. Conecte seu computador à porta USB2 .0 FS no RSK+ para N-2MB. RX65
3. Conecte seu computador à USB-to-serial porta no RSK+ para RX65 N-2MB.
4. Conecte um roteador ou porta Ethernet conectada à Internet à porta Ethernet no RSK+ para N-2MB. RX65

Para configurar o módulo depurador E2 Lite

1. Use o cabo de fita de 14 pinos para conectar o módulo E2 Lite Debugger à porta 'E1/E2 Lite' no RSK+ para N-2MB. RX65
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 RX65 N-2MB, os drivers do depurador E2 Lite começarão a ser instalados.

   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+ RX65 para N-2MB, use o IDE de estúdio Renesas e 2 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.

Execução do projeto FreeRTOS

Para executar o projeto no e^{2}studio

1. Confirme se você conectou o módulo E2 Lite Debugger ao seu RSK+ para N-2MB RX65
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.

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

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 STM32 L4 Discovery Kit IoT Node

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o STMicroelectronics STM32 L4 Discovery Kit IoT Node. Se você ainda não tem o STMicroelectronics STM32 L4 Discovery Kit IoT Node, visite AWS o Catálogo de dispositivos do 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 Kit STM32 L4 Discovery, nó IoT, conexão sem fio de baixa potência, Bluetooth Low Energy, NFC, Sub, Wi-Fi. GHz Em Binary Resources (Recursos binários), escolha Inventek ISM 43362 Wi-Fi module firmware update (read the readme file for instructions) (Atualização de firmware do módulo Inventek ISM 43362 Wi-Fi (ler o arquivo readme para obter instruções)) .

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. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilar uma aplicação 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.

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.

Compilação e execução do projeto de demonstração do FreeRTOS

Importe a demonstração do FreeRTOS para o System Workbench STM32

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.

Execução do projeto de demonstração do FreeRTOS

1. Use um cabo USB para conectar seu STMicroelectronics STM32 L4 Discovery Kit IoT Node ao seu computador.
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 STM32 Depuração 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 FreeRTOS

Se você preferir não usar um IDE para o desenvolvimento de Freertos, você pode CMake usar como alternativa para criar e executar os aplicativos de demonstração ou os 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 o uso CMake com FreeRTOS, consulte. Usando CMake com FreeRTOS

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*/example/topic** e selecione Inscreva-se no tópico.

Solução de problemas

Se você vir o seguinte na saída UART da aplicação 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 Kit STM32 L4 Discovery, nó IoT, conexão sem fio de baixa potência, Bluetooth Low Energy, NFC, Sub, Wi-Fi. GHz Em Binary Resources (Recursos binários), escolha o link de download de Inventek ISM 43362 Wi-Fi module firmware update (Atualização de firmware do módulo Inventek ISM 43362 Wi-Fi).

Para obter mais informações sobre soluções de problemas gerais sobre os Conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Começando com a Texas Instruments CC322 0SF-LAUNCHXL

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o Texas Instruments CC322 0SF-LAUNCHXL. Se você não tiver o kit de desenvolvimento CC322 0SF-LAUNCHXL 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 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. Instalar software na máquina host para desenvolver e depurar aplicativos incorporados para seu microcontrolador.
2. Compilar uma aplicação 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.

Configuração do ambiente de desenvolvimento

Siga as etapas abaixo para configurar o ambiente de desenvolvimento para começar a usar o FreeRTOS.

Observe que o FreeRTOS suporta IDEs dois para o kit de desenvolvimento CC322 TI 0SF-LAUNCHXL: Code Composer Studio e IAR Embedded Workbench versão 8.32. Você pode usar o 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 Troubleshooting CCS.

Instalar o IAR Embedded Workbench

1. Faça download e execute o Instalador do Windows para a versão 8.32 do IAR Embedded Workbench for ARM. Em Debug probe drivers (Depurar drivers de teste), selecione TI XDS.
2. Conclua a instalação e inicie o programa. Na página License Wizard (Assistente de licença), escolha Register with IAR Systems to get an evaluation license (Registrar com IAR Systems para obter uma licença de avaliação) ou use sua própria licença do IRA.

Instale o SDK SimpleLink CC322 0

Instale o SDK SimpleLink CC322 0. O SDK SimpleLink Wi-Fi CC322 0 contém drivers para o MCU programável CC322 0SF, mais de 40 aplicativos de amostra e a documentação necessária para usar as amostras.

Instalar o Uniflash

Instale o Uniflash. O CCS Uniflash é uma ferramenta autônoma usada para programar a memória flash no chip na TI. MCUs O Uniflash tem uma GUI, linha de comando e interface de scripts.

Instalação do Service Pack mais recente

1. Em seu TI CC322 0SF-LAUNCHXL, coloque o jumper SOP 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-LAUNCHXL na lista de placas em Nova configuração e, em seguida, escolha Iniciar Criador de Imagem.
3. Escolha 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 SDK CC322 0SF SimpleLink . 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 alternar o jumper SOP de volta para a posição 0 e redefina a placa.

Configuração de provisionamento de Wi-Fi

Para definir as configurações de Wi-Fi para sua placa, execute uma das seguintes ações:

- Configure a aplicação de demonstração do FreeRTOS descrita em Configuração das demonstrações do FreeRTOS.
- Use SmartConfigda Texas Instruments.

Compilação e execução do projeto de demonstração do FreeRTOS

Compilação e execução do projeto de demonstração do FreeRTOS no TI Code Composer

Como importar a demonstração do FreeRTOS 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.

Como executar a demonstração do FreeRTOS 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 cabo USB para conectar o 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).

Criação e execução do projeto de demonstração do FreeRTOS no IAR Embedded Workbench

Para importar a demonstração do FreeRTOS para o IAR Embedded Workbench

1. Abra o IAR Embedded Workbench, escolha File (Arquivo) e, em seguida, escolha Open Workspace (Abrir 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).

Como executar a demonstração do FreeRTOS 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 cabo USB para conectar o 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 EnergyTrace Tecnologia MSP.
5. Quando o depurador parar no ponto de interrupção em `main()`, vá para o menu Debug (Depurar) e escolha Go (Ir).

Usando CMake com FreeRTOS

Se você preferir não usar um IDE para o desenvolvimento de Freertos, você pode CMake usar como alternativa para criar e executar os aplicativos de demonstração ou os aplicativos que você desenvolveu usando editores de código e ferramentas de depuração de terceiros.

Para criar a demonstração do FreeRTOS com CMake

1. Crie uma pasta para conter os arquivos de compilação gerados (*`build-folder`*).
2. Verifique se o caminho de pesquisa (variável de ambiente $PATH) contém a pasta onde o binário do compilador TI CGT está localizado (por exemplo, `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`).

   Se você estiver usando o compilador de TI ARM com sua placa TI, use o comando a seguir para gerar arquivos de compilação do código-fonte:

   ```
   cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S freertos -B build-folder
   ```

Para obter mais informações, consulte Usando CMake com FreeRTOS.

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***`your-thing-name`*`/example/topic`** e selecione Inscreva-se no tópico.

Solução de problemas

Se você não vê mensagens no cliente MQTT 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 mais informações sobre soluções de problemas gerais sobre os Conceitos básicos do FreeRTOS, consulte Solução de problemas de conceitos básicos.

Conceitos básicos do simulador de dispositivo do Windows

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tutorial fornece instruções para começar a usar o simulador de dispositivos do Windows com o FreeRTOS.

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`*.

O FreeRTOS é liberado como um arquivo zip que contém as bibliotecas e aplicações 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 sucesso em instâncias EC2 do Amazon Windows.

Configuração do ambiente de desenvolvimento

1. Instale a versão mais recente do Win PCap.
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 CMake baseado para criar seus projetos FreeRTOS, instale a versão mais recente do. CMake O FreeRTOS CMake requer a versão 3.13 ou posterior.

Compilação e execução do projeto de demonstração do FreeRTOS

Você pode usar o Visual Studio ou CMake criar 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.

Construindo e executando o projeto de demonstração do FreeRTOS com CMake

Recomendamos que você use a CMake GUI em vez da ferramenta de CMake linha de comando para criar o projeto de demonstração para o Simulador do Windows.

Depois de instalar CMake, abra a CMake GUI. No Windows, você pode encontrar isso no menu Iniciar CMake, em 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 CMake projeto.

   Na CMake GUI, escolha Adicionar entrada e, na janela Adicionar entrada de cache, defina os seguintes valores:

   Name

   AFR_BOARD

Tipo

STRING

Valor

pc.windows

Descrição

(Opcional)

3. Selecione Configurar. Se for CMake solicitado que você crie o diretório de compilação, escolha Sim e selecione um gerador em Especificar o gerador para este 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.) Deixe as outras opções de gerador inalteradas e escolha Finish (Concluir).
4. Gere e abra o CMake projeto.

   Depois de configurar o projeto, a CMake GUI mostra todas as opções disponíveis para o projeto gerado. Para os fins deste tutorial, você pode deixar as opções com 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 `aws_demos` projeto e escolha Definir como StartUp projeto. 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 CMake com FreeRTOS, consulte. Usando CMake com 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 enumera as 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 da aplicação. No exemplo acima, 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. Para obter mais informações, consulte a entrada de WinPcap perguntas frequentes Q-16: Quais adaptadores de rede são compatíveis? WinPcap .

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*/example/topic** e selecione Inscreva-se no tópico.

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.

Começando com o kit de IoT industrial Xilinx Avnet MicroZed

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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

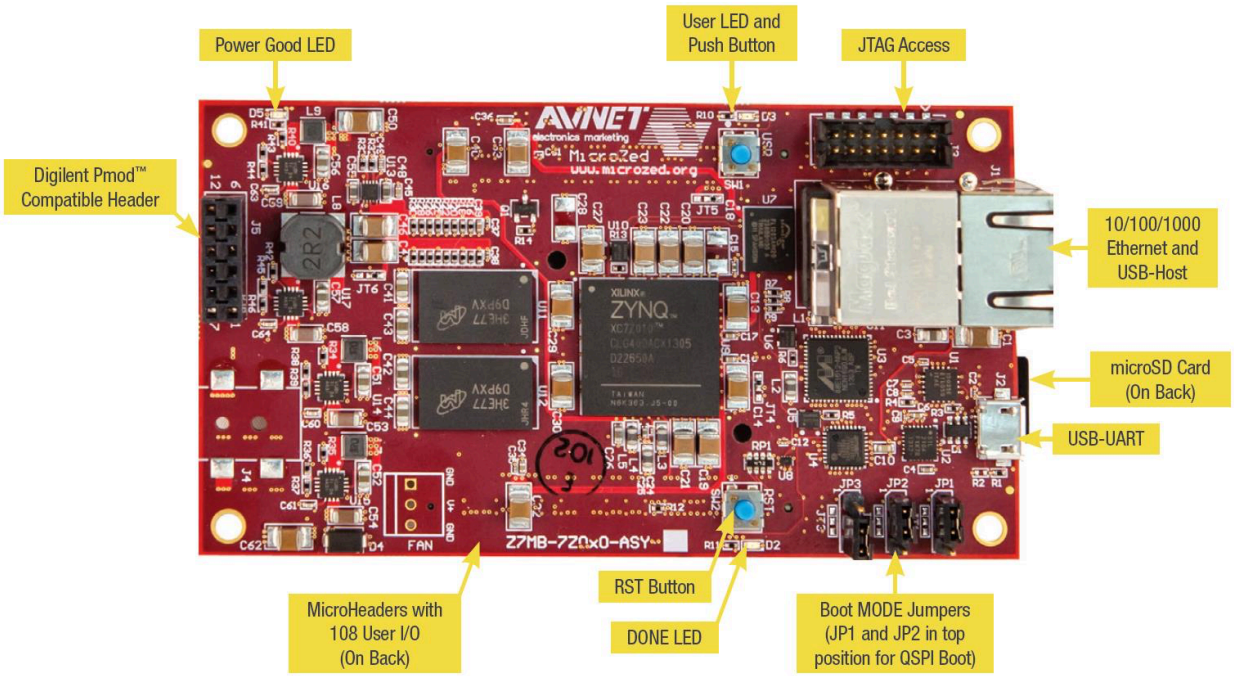
Este tutorial contém instruções para as seguintes etapas iniciais:

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 seu computador não detectar a MicroZed conexão USB-UART, instale os drivers CP21 0x USB-to-UART Bridge VCP manualmente. Para obter instruções, consulte o Guia de USB-to-UART instalação do Silicon Labs CP21 0x.

Para obter mais informações sobre o XSDK, consulte Conceitos básicos do Xilink SDK no site do Xilink.

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`.

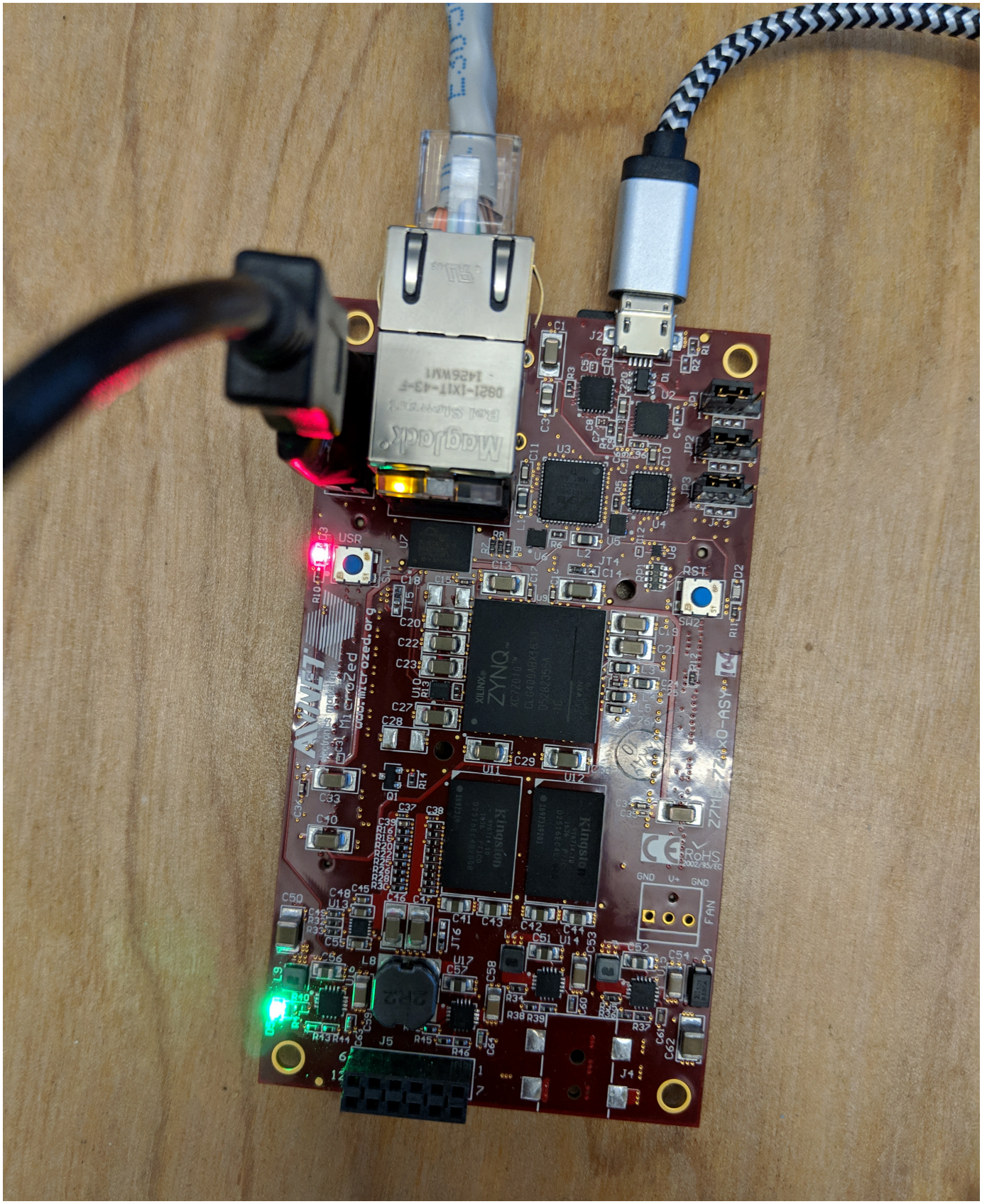
Depuração do JTAG

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
ZYNQ
XILINX
Kingston
GND
V+
GND
FAN
RoHS

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

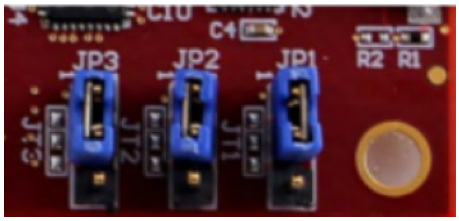
Formate o cartão microSD fornecido com o kit de IoT industrial Xilinx MicroZed.

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.

Monitoramento de mensagens MQTT na nuvem

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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***your-thing-name*****/example/topic** e selecione Inscreva-se no tópico.

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.

Atualizações do FreeRTOS Over-the-Air

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Over-the-air As atualizações (OTA) permitem que você implante atualizações de firmware em um ou mais dispositivos em sua 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. A aplicação de demonstração OTA é executada 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 na 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 a aplicação de demonstração de atualização OTA do FreeRTOS, deverá integrar o Biblioteca do agente OTA em sua própria aplicação para obter o recurso de atualização de firmware.

As atualizações do over-the-air FreeRTOS possibilitam que você:

- 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.

- Verificar 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 permitem que você categorize seus AWS IoT recursos de maneiras diferentes (por exemplo, por finalidade, proprietário ou 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Para usar as 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 a assinatura de código para AWS IoT,Conceda acesso à assinatura de código para AWS IoT.
- Faça download do FreeRTOS com a biblioteca OTA.

Criação de um bucket do Amazon S3 para armazenar a atualização

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Os arquivos de atualização OTA são armazenados em buckets do Amazon S3.

Se você estiver usando a Assinatura de Código para AWS IoT, o comando usado para criar um trabalho de assinatura de código usa 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 dos arquivos são alterados para GUIDs que os arquivos originais não sejam sobrescritos.

Como criar um bucket do Amazon S3

1. Faça login no console do Amazon S3 em. https://console.aws.amazon.com/s3/
2. Escolha 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. Escolha 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 no https://console.aws.amazon.com/iam/.
2. No painel de navegação, escolha Roles (Funções).
3. Selecione Criar perfil.
4. Em Select type of trusted entity (Selecionar tipo de entidade confiável), selecione AWS Serviço.
5. Escolha IoT na lista de serviços. AWS
6. Em Select your use case (Selecione seu caso de uso), escolha IoT.
7. Escolha Próximo: Permissões.
8. Escolha Next: Tags (Próximo: tags).
9. Selecione Próximo: revisar.
10. Digite um nome e uma descrição de função e escolha Create role (Criar função).

Para obter mais informações sobre funções do IAM, consulte Funções do IAM.

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 Anexar políticas.
3. Na caixa Pesquisar, digite "AmazonFreeRTOSOTAUpdate“, selecione na lista `AmazonFreeRTOSOTAUpdate` de políticas filtradas e escolha Anexar política para anexar a política à sua 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. Selecione 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 *your_account_id* substituir pelo ID AWS da sua conta e *your_role_name* pelo nome da função de serviço OTA.
5. Selecione 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 AWS gerenciada `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. Selecione 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 *example-bucket* substituir pelo nome do seu bucket.

5. Selecione Revisar política.
6. Insira um nome para a política e escolha Create policy (Criar política).

Criação de uma política de usuário de OTA

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Você deve conceder permissão ao usuário do IAM para realizar over-the-air atualizações. O usuário deve ter permissões para:

- Acessar o bucket do S3 onde suas atualizações de firmware estão armazenadas.
- Certificados de acesso armazenados em AWS Certificate Manager.

- Acesse o recurso de entrega de arquivos AWS IoT baseado em MQTT.
- Acessar as atualizações OTA do FreeRTOS.
- Acesse AWS IoT empregos.
- Acessar o IAM.
- Assinatura de código de acesso para AWS IoT. Consulte Conceda acesso à assinatura de código para AWS IoT.
- Listar as plataformas de hardware FreeRTOS.
- Marque e desmarque AWS IoT recursos.

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.

Criação de um certificado de assinatura de código

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 (AC) 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 0SF-LAUNCHXL CC322
- Criando um certificado de assinatura de código para o Microchip Curiosity MZEF PIC32
- 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 0SF-LAUNCHXL CC322

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O kit de desenvolvimento do microcontrolador sem fio SimpleLink Wi-Fi CC322 0SF Launchpad suporta 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 do playground permite experimentar as atualizações OTA com um certificado de assinatura de código autoassinado.

Use o AWS Command Line Interface para importar seu certificado de assinatura de código, chave privada e cadeia de certificados para. AWS Certificate Manager Para obter mais informações, consulte Instalar a AWS CLI no Guia do usuário da AWS Command Line Interface .

Baixe e instale a versão mais recente do SimpleLink CC3220 SDK. 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 SDK SimpleLink CC322 0 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 vinculado à hierarquia de certificados do playground da Texas Instruments e que AWS Certificate Manager atenda aos critérios de Assinatura de Código. 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`. *`test_signer@amazon.com`*Substitua 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 para: 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**
>
> Esta etapa foi escrita com a suposição de que você usará a Assinatura de Código AWS IoT para assinar suas imagens de firmware. Embora o uso do Code Signing AWS IoT for seja recomendado, você pode assinar suas imagens de firmware manualmente.

Criando um certificado de assinatura de código para o Microchip Curiosity MZEF PIC32

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O Microchip Curiosity PIC32 MZEF suporta um certificado autoassinado SHA256 com 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 o AWS Command Line Interface para importar seu certificado de assinatura de código, chave privada e cadeia de certificados para. AWS Certificate Manager Para obter informações sobre a instalação do AWS CLI, consulte Instalando AWS CLI o.

1. No diretório de trabalho, use o texto a seguir para criar um arquivo chamado `cert_config.txt`. *`test_signer@amazon.com`*Substitua 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 para: 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**
>
> Esta etapa foi escrita com a suposição de que você usará a Assinatura de Código AWS IoT para assinar suas imagens de firmware. Embora o uso do Code Signing AWS IoT for seja recomendado, você pode assinar suas imagens de firmware manualmente.

Criação de um certificado de assinatura de código para o Espressif ESP32

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

As ESP32 placas Espressif suportam 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 o AWS Command Line Interface para importar seu certificado de assinatura de código, chave privada e cadeia de certificados para. AWS Certificate Manager Para obter informações sobre a instalação do AWS CLI, consulte Instalando AWS CLI o.

1. No diretório de trabalho, use o texto a seguir para criar um arquivo chamado `cert_config.txt`. *`test_signer@amazon.com`*Substitua 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 para: 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**
>
> Esta etapa foi escrita com a suposição de que você usará a Assinatura de Código AWS IoT para assinar suas imagens de firmware. Embora o uso do Code Signing AWS IoT for seja recomendado, você pode assinar suas imagens de firmware manualmente.

Criação de um certificado de assinatura de código para Nordic nrf52840-dk

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O Nordic nrf52840-dk suporta um certificado autoassinado com assinatura de código ECDSA. SHA256

> **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 o AWS Command Line Interface para importar seu certificado de assinatura de código, chave privada e cadeia de certificados para. AWS Certificate Manager Para obter informações sobre a instalação do AWS CLI, consulte Instalando AWS CLI o.

1. No diretório de trabalho, use o texto a seguir para criar um arquivo chamado `cert_config.txt`. *`test_signer@amazon.com`*Substitua 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 para: 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**
>
> Esta etapa foi escrita com a suposição de que você usará a Assinatura de Código AWS IoT para assinar suas imagens de firmware. Embora o uso do Code Signing AWS IoT for seja recomendado, você pode assinar suas imagens de firmware manualmente.

Criação de um certificado de assinatura de código para o simulador do Windows do FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário 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 o AWS Command Line Interface para importar seu certificado de assinatura de código, chave privada e cadeia de certificados para. AWS Certificate Manager Para obter informações sobre a instalação do AWS CLI, consulte Instalando AWS CLI o.

1. No diretório de trabalho, use o texto a seguir para criar um arquivo chamado `cert_config.txt`. *`test_signer@amazon.com`*Substitua 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 para: 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**
>
> Esta etapa foi escrita com a suposição de que você usará a Assinatura de Código AWS IoT para assinar suas imagens de firmware. Embora o uso do Code Signing AWS IoT for seja recomendado, você pode assinar suas imagens de firmware manualmente.

Criação de um certificado de assinatura de código para o hardware personalizado

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Usando um conjunto de ferramentas apropriado, crie um certificado autoassinado e uma chave privada para o hardware.

Use o AWS Command Line Interface para importar seu certificado de assinatura de código, chave privada e cadeia de certificados para. AWS Certificate Manager Para obter informações sobre a instalação do AWS CLI, consulte Instalando AWS CLI o.

Depois de criar seu certificado de assinatura de código, você pode usar o 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 `aws_ota_codesigner_certificate.h` que faz parte do código do FreeRTOS que você obterá por download posteriormente.

Conceda acesso à assinatura de código para AWS IoT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Em ambientes de produção, você deve assinar digitalmente sua atualização de firmware para garantir a autenticidade e a integridade da atualização. Você pode assinar sua atualização manualmente ou usar a Assinatura de Código AWS IoT para assinar seu código. Para usar a assinatura de código do FreeRTOS, é necessário conceder à sua conta de usuário do IAM acesso à assinatura de código do FreeRTOS.

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.

Faça download do FreeRTOS com a biblioteca OTA

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para uso com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Você pode clonar ou baixar FreeRTOS em. GitHub Consulte o arquivo README.md para obter instruções.

Para obter informações sobre como configurar e executar a aplicação de demonstração OTA, consulte Over-the-air atualiza o aplicativo de demonstração.

Pré-requisitos para atualizações de OTA usando MQTT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Esta seção descreve os requisitos gerais para o uso do MQTT para executar over-the-air (atualizações OTA).

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.

Requisitos mínimos

- O firmware do dispositivo deve incluir as bibliotecas do FreeRTOS necessárias (coreMQTT, agente 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 2019.12.00, o FreeRTOS OTA pode usar o protocolo HTTP ou MQTT para transferir imagens de atualização de firmware para dispositivos. AWS IoT 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 `aws_ota_agent_config.h` é usar o protocolo MQTT:

```
/**
 * @brief The protocol selected for OTA control operations.
 * This configuration parameter sets the default protocol for all the OTA control
 * operations like requesting OTA job, updating the job status etc.
 *
 * Note - Only MQTT is supported at this time for control operations.
 */
#define configENABLED_CONTROL_PROTOCOL       ( OTA_CONTROL_OVER_MQTT )
/**
 * @brief The protocol selected for OTA data operations.
 * This configuration parameter sets the protocols selected for the data operations
 * like requesting file blocks from the service.
 *
 * Note - Both MQTT and HTTP are supported for data transfer. This configuration
 parameter
 * can be set to the following -
 * Enable data over MQTT - ( OTA_DATA_OVER_MQTT )
 * Enable data over HTTP - ( OTA_DATA_OVER_HTTP)
 * Enable data over both MQTT & HTTP ( OTA_DATA_OVER_MQTT | OTA_DATA_OVER_HTTP )
 */
#define configENABLED_DATA_PROTOCOLS         ( OTA_DATA_OVER_MQTT )
 /**
  * @brief The preferred protocol selected for OTA data operations.
  *
  * Primary data protocol will be the protocol used for downloading files if more than
  * one protocol is selected while creating OTA job. Default primary data protocol is
 MQTT
  * and the following update here switches to HTTP as primary.
  *
  * Note - use OTA_DATA_OVER_HTTP for HTTP as primary data protocol.
  */
#define configOTA_PRIMARY_DATA_PROTOCOL     ( OTA_DATA_OVER_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 o MQTT deve ser registrado como um item AWS IoT e 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 `iot:Connect` permissões permitem que seu dispositivo se conecte AWS IoT ao MQTT.
- As `iot:Publish` permissões `iot:Subscribe` e sobre os tópicos de AWS IoT trabalhos (`.../jobs/*`) permitem que o dispositivo conectado receba notificações 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.

Pré-requisitos para atualizações de OTA usando HTTP

Important

Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Esta seção descreve os requisitos gerais para o uso de HTTP para realizar atualizações over-the-air (OTA). A partir da versão 2019.12.00, o FreeRTOS OTA pode usar o protocolo HTTP ou MQTT para transferir imagens de atualização de firmware para dispositivos. AWS IoT

Note

- Embora o protocolo HTTP possa ser usado para transferir a imagem do firmware, a biblioteca coreMQTT ainda é necessária porque outras interações com o AWS IoT Core usam a biblioteca coreMQTT, 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 (coreMQTT, HTTP, agente OTA e suas dependências).
- A versão 201912.00 ou posterior o FreeRTOS é necessária para alterar a configuração de protocolos de 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\aws_ota_agent_config.h`.

```
/**
 * @brief The protocol selected for OTA control operations.
 * This configuration parameter sets the default protocol for all the OTA control
 * operations like requesting an OTA job, updating the job status, and so on.
 *
 * Note - Only MQTT is supported at this time for control operations.
 */
#define configENABLED_CONTROL_PROTOCOL       ( OTA_CONTROL_OVER_MQTT )
/**
 * @brief The protocol selected for OTA data operations.
 * This configuration parameter sets the protocols selected for the data operations
 * like requesting file blocks from the service.
 *
 * Note - Both MQTT and HTTP are supported for data transfer. This configuration
 parameter
 * can be set to the following -
 * Enable data over MQTT - ( OTA_DATA_OVER_MQTT )
 * Enable data over HTTP - ( OTA_DATA_OVER_HTTP)
 * Enable data over both MQTT & HTTP ( OTA_DATA_OVER_MQTT | OTA_DATA_OVER_HTTP )
 */
#define configENABLED_DATA_PROTOCOLS         ( OTA_DATA_OVER_MQTT )
 /**
  * @brief The preferred protocol selected for OTA data operations.
  *
  * Primary data protocol will be the protocol used for downloading files if more than
```

```
  * one protocol is selected while creating OTA job. Default primary data protocol is
 MQTT
  * and the following update here switches to HTTP as primary.
  *
  * Note - use OTA_DATA_OVER_HTTP for HTTP as primary data protocol.
  */
#define configOTA_PRIMARY_DATA_PROTOCOL     ( OTA_DATA_OVER_MQTT )
```

Para habilitar a transferência de dados OTA via HTTP

1. Fazer `configENABLED_DATA_PROTOCOLS` alteração `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 CC322 0SF-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 PIC32 MZEF

- Operações de controle (MQTT): 65 KB
- Operações de dados (HTTP): 43 KB

Expressivo ESP32

- Operações de controle (MQTT): 65 KB
- Operações de dados (HTTP): 45 KB

> **Note**
>
> BLE on ESP32 ocupa cerca de 87 KB de 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 `iot:Publish` permissões `iot:Subscribe` e sobre os tópicos de AWS IoT trabalhos (`.../jobs/*`) permitem que o dispositivo conectado receba notificações e documentos de trabalho e publique o estado de conclusão da execução de um trabalho.
- As `iot:Receive` permissões AWS IoT Core permitem publicar mensagens sobre esses tópicos no dispositivo conectado atual. 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 AWS IoT console ou o 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. Isso AWS CLI é útil quando você está automatizando trabalhos de atualização do OTA, trabalhando com um grande número de dispositivos ou usando dispositivos que não foram qualificados para 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 a assinatura de código para AWS IoT, faça o upload da sua 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 pela aplicação, 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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

- Instale a versão inicial do firmware no Texas Instruments CC322 0SF-LAUNCHXL
- Instale a versão inicial do firmware no Microchip Curiosity PIC32 MZEF
- Instale a versão inicial do firmware no Espressif ESP32
- Instale a versão inicial do firmware no Nordic n RF5284 0 DK
- Firmware inicial no simulador do Windows
- Instalação da versão inicial do firmware em uma placa personalizada

Instale a versão inicial do firmware no Texas Instruments CC322 0SF-LAUNCHXL

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Essas etapas foram escritas presumindo que você já criou o projeto `aws_demos`, conforme descrito em Baixe, crie, atualize e execute a demonstração do FreeRTOS OTA no Texas Instruments 0SF-LAUNCHXL CC322.

1. Em seu Texas Instruments CC322 0SF-LAUNCHXL, coloque o jumper SOP no meio do conjunto de pinos (posição = 1) e reinicie a placa.
2. Faça download e instale a ferramenta TI Uniflash.
3. Inicie o Uniflash. Na lista de configurações, escolha CC3220SF-LAUNCHXL e, em seguida, escolha Start Image Creator.
4. Escolha Novo projeto.
5. 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). Escolha Criar projeto.
6. Desconecte o emulador de terminal.
7. No lado direito da janela da aplicação 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 Open (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 Arquivo de origem, escolha Procurar, escolha ***version***simplelink_cc32xx_sdk_/20160911.lst e escolha Abrir. tools/cc32xx_tools/certificate-playground/certcatalogPlayGround
19. Em Arquivo de origem da assinatura, escolha Procurar, escolha simplelink_cc32xx_sdk_/20160911.lst.signed_3220.bin ***version*** e escolha Abrir. tools/cc32xx_tools/certificate-playground/certcatalogPlayGround
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 da aplicação 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
```

Instale a versão inicial do firmware no Microchip Curiosity PIC32 MZEF

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Essas etapas foram escritas presumindo que você já criou o projeto `aws_demos`, conforme descrito em Baixe, crie, atualize e execute a demonstração do FreeRTOS OTA no Microchip Curiosity MZEF PIC32.

Para gravar a aplicação de demonstração na placa

1. Recrie o projeto (`aws_demos`) e certifique-se de que ele seja compilado sem erros.
2. Na barra de ferramentas, escolha

   .
3. Depois que o processo de programação estiver concluído, desconecte o depurador do ICD 4 e reinicie a placa. Reconecte seu emulador de terminal para garantir que a saída seja a mesma de quando você depurou a demonstração com o MPLAB X IDE.

   O terminal deve exibir uma saída semelhante à seguinte.

```
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 procedimento a seguir cria um arquivo hexadecimal unificado ou uma imagem de fábrica que consiste em um bootloader de referência e uma aplicação com uma assinatura criptográfica. O bootloader verifica a assinatura criptográfica da aplicação na inicialização e oferece suporte a atualizações OTA.

Para criar e atualizar uma imagem de fábrica

1. Verifique se você tem as SRecord ferramentas instaladas do Source Forge. Verifique se o diretório que contém os programas `srec_info` e `srec_cat` está no caminho do sistema.
2. Atualize o número de sequência OTA e a versão da aplicação para a imagem de fábrica.
3. Crie o projeto `aws_demos`.
4. Execute o script `factory_image_generator.py` para gerar a imagem de fábrica.

   ```
   factory_image_generator.py -b mplab.production.bin -p MCHP-Curiosity-PIC32MZEF -k
    private_key.pem  -x aws_bootloader.X.production.hex
   ```

   Esse comando usa os seguintes parâmetros:

   - `mplab.production.bin`: o binário da aplicação.
   - `MCHP-Curiosity-PIC32MZEF`: o nome da plataforma.
   - `private_key.pem`: a chave privada de assinatura de código.
   - `aws_bootloader.X.production.hex`: o arquivo hexadecimal do bootloader.

Quando você cria o projeto `aws_demos`, a imagem binária da aplicação e o arquivo hexadecimal do bootloader são criados como parte do processo. Cada projeto no diretório `vendors/microchip/boards/curiosity_pic32mzef/aws_demos/` contém um diretório `dist/pic32mz_ef_curiosity/production/` que contém esses arquivos. O arquivo hexadecimal unificado gerado é denominado `mplab.production.factory.unified.hex`.

5. Use a ferramenta MPLab IPE para programar o arquivo hexadecimal gerado no dispositivo.
6. Você pode verificar se a imagem de fábrica funciona observando a saída UART da placa enquanto é feito o upload da imagem. Se tudo estiver configurado corretamente, você verá a imagem inicializar corretamente:

```
[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] Valid magic code at: 0xbd000000
[prvValidateImage] Valid image flags: 0xfc at: 0xbd000000
[prvValidateImage] Addresses are valid.
[prvValidateImage] Crypto signature is valid.
[...]
[prvBOOT_ValidateImages] Booting image with sequence number 1 at 0xbd000000
```

7. Se os certificados estiverem configurados incorretamente ou se uma imagem OTA não estiver assinada corretamente, você poderá ver mensagens como as seguintes antes que o bootloader do chip apague a atualização inválida. Verifique se os certificados de assinatura de código estão consistentes e revise as etapas anteriores com atenção.

```
[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] Valid magic code at: 0xbd000000
[prvValidateImage] Valid image flags: 0xfc at: 0xbd000000
[prvValidateImage] Addresses are valid.
[prvValidateImage] Crypto signature is not valid.
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000
[BOOT_FLASH_EraseBank] Bank erased at : 0xbd000000
```

Instale a versão inicial do firmware no Espressif ESP32

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este guia foi escrito partindo do pressuposto de que você já executou as etapas em Introdução aos pré-requisitos do Espressif ESP32 - DevKit C e de atualização. ESP-WROVER-KIT Over-the-Air 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_MQTT_DEMO_ENABLED` e defina `CONFIG_OTA_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 demos/include/aws_ota_codesigner_certificate.h. Ele deve ser formatado como a seguir.

   ```
   static const char signingcredentialSIGNING_CERTIFICATE_PEM[] = "-----BEGIN
    CERTIFICATE-----\n"
   "...base64 data...\n"
   "-----END CERTIFICATE-----\n";
   ```

3. Com a demonstração da atualização OTA selecionada, siga as mesmas etapas descritas em Introdução 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 a aplicação 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 ESP32 diretoria agora está ouvindo as atualizações do 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.

Instale a versão inicial do firmware no Nordic n RF5284 0 DK

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este guia foi escrito com a suposição de que você já executou as etapas Começando com o Nordic n 0-DK RF5284 e Over-the-Air atualizou os pré-requisitos. 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. Abra o *freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h.
2. Substitua `#define CONFIG_MQTT_DEMO_ENABLED` por `#define democonfigOTA_UPDATE_DEMO_ENABLED`.
3. Com a demonstração da atualização OTA selecionada, siga as mesmas etapas descritas em Começando com o Nordic n 0-DK RF5284 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Quando você usa o simulador do Windows, não há necessidade de atualizar uma versão inicial do firmware. O simulador do Windows é parte da aplicação `aws_demos`, que também inclui o firmware.

Instalação da versão inicial do firmware em uma placa personalizada

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 a aplicação na placa e verifique se ela está sendo executada corretamente.

Atualização da versão do firmware

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 da aplicação de demonstração OTA.

1. Abra o projeto `aws_demos` no IDE.
2. Abra `demos/include/aws_application_version.h` e incremente o valor do token `APP_VERSION_BUILD`.
3. Se você estiver usando o Microchip Curiosity PIC32 MZEF, incremente o número de sequência OTA em. `vendors/microchip/boards/curiosity_pic32mzef/bootloader/bootloader/utility/user-config/ota-descriptor.config` O número de sequência OTA deve ser incrementado para cada nova imagem OTA gerada.
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 CC322 0SF-LAUNCHXL: `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- Microchip Curiosity PIC32 MZEF: `vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/dist/pic32mz_ef_curiosity/production/mplab.production.ota.bin`
- Expressivo ESP32: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

Criando uma atualização OTA (AWS IoT console)

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

1. No painel de navegação do AWS IoT console, escolha Gerenciar e, em seguida, escolha Trabalhos.

2. Escolha Criar.
3. Em Criar um trabalho de atualização do Over-the-Air FreeRTOS (OTA), escolha Criar trabalho de atualização OTA.
4. Você pode implantar uma atualização OTA em um único dispositivo ou em um grupo de dispositivos. Em Select devices to update (Selecionar dispositivos para atualizar), escolha Select (Selecionar). Para atualizar um único dispositivo, escolha a guia Things (Coisas). Para atualizar um grupo de dispositivos, escolha a guia Thing Groups (Grupos de coisas).
5. Se você estiver atualizando um único dispositivo, marque a caixa de seleção ao lado da coisa da IoT associada ao dispositivo. Se você estiver atualizando um grupo de dispositivos, marque a caixa de seleção ao lado do grupo de coisas associado aos dispositivos. Escolha Próximo.
6. Em Select the protocol for firmware image transfer (Selecione o protocolo para transferência de imagem de firmware), escolha HTTP, MQTT ou ambos para permitir que cada dispositivo determine o protocolo a ser usado.
7. Em Select and sign your firmware image (Selecionar e assinar a imagem de firmware), escolha Sign a new firmware image for me (Assinar uma nova imagem de firmware para mim).
8. Em Code signing profile (Perfil de assinatura de código), escolha Create (Criar).
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 você estiver usando uma plataforma não qualificada e estiver usando um pacote de criptografia diferente do ECDSA P-256 SHA-256 para assinatura, você pode usar a Assinatura de Código para AWS IoT, ou você mesmo pode assinar a atualização do firmware. Para obter mais informações, consulte Assinatura digital da atualização de firmware.

b. Em Code signing certificate (Certificado de assinatura de código), escolha Select (Selecionar) para selecionar um certificado importado anteriormente ou Import (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. A localização do certificado varia de acordo com a plataforma. Deve ser o local onde você colocou o certificado de assinatura de código quando seguiu as instruções em Instalação do firmware inicial.

> **Important**
>
> No Texas Instruments CC322 0SF-LAUNCHXL, não inclua uma barra inicial (**/**) na frente do nome do arquivo se seu 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`.

10. Em Select your firmware image in S3 or upload it (Selecionar a imagem de firmware no S3 ou fazer upload dela), escolha Select (Selecionar). 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 PIC32 MZEF produzem duas imagens binárias com nomes padrão de e. `mplab.production.bin` `mplab.production.ota.bin` Use o segundo arquivo ao fazer upload de uma imagem para a atualização OTA.

11. 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.

> **Important**
>
> No Texas Instruments CC322 0SF-LAUNCHXL, devido a restrições de segurança, o nome do caminho da imagem do firmware deve ser. `/sys/mcuflashimg.bin`

12. Em Tipo de arquivo, 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 a qual processador a atualização se destina.
13. Em IAM role for OTA update job (Função do IAM para trabalho de atualização OTA), escolha uma função de acordo com as instruções em Criação de uma função de serviço de atualização de OTA.
14. Escolha Próximo.
15. Insira um ID e uma descrição para seu trabalho de atualização OTA.
16. Em Tipo de trabalho, escolha Seu trabalho será concluído após a implantação no selecionado devices/groups (instantâneo).
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 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. Esse nome de caminho varia por plataforma.
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 aparece no AWS IoT console com o status EM ANDAMENTO.

> **Note**
>
> - O AWS IoT console 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 Over-the-air atualiza o aplicativo de demonstração.

Criando uma atualização OTA com o AWS CLI

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Ao usar o 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 o AWS CLI para realizar atualizações OTA, você pode usar a Assinatura de Código para AWS IoT, ou você mesmo pode assinar sua atualização de firmware. Para obter uma lista dos

algoritmos de assinatura criptográfica e hashing suportados pelo Code Signing for AWS IoT, consulte. SigningConfigurationOverrides Se você quiser usar um algoritmo criptográfico que não é suportado pelo Code Signing for AWS IoT, você deve assinar o binário do seu firmware antes de carregá-lo no Amazon S3.

Assinando sua imagem de firmware com Code Signing for AWS IoT

Para assinar sua imagem de firmware usando o Code Signing for AWS IoT, você pode usar uma das ferramentas de linha de comando AWS SDKs ou. Para obter mais informações sobre Assinatura de código para AWS IoT, consulte Assinatura de código para AWS IoT.

Depois de instalar e configurar as ferramentas de assinatura de código, copie sua imagem de firmware não assinada para o bucket do Amazon S3 e inicie um trabalho de assinatura de código com os seguintes comandos. 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* pode 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 Assinatura de código para AWS IoT, consulte Assinatura de código 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 AWS IoT OTA, você também pode criar um stream que contém sua 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 a Assinatura de Código para AWS IoT, o nome do arquivo é um GUID gerado pela Assinatura de Código para AWS IoT.

Use o comando create-stream 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 create-stream AWS CLI comando:

`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 a Assinatura de Código 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 sua imagem de firmware assinada, use o aws signer describe-signing-job --job-id ***`my-job-id`*** comando where `my-job-id` is the job ID exibido pelo create-signing-job AWS CLI comando. 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 create-ota-update AWS CLI comando 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`**

Name	Tipo	Descrição
`otaUpdateId`	string (máx: 128 mín:1)	O ID da atualização OTA a ser criado.
`description`	string (máx: 2028)	A descrição da atualização OTA.
`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.
`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

Name	Tipo	Descrição
		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.
`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.

Name	Tipo	Descrição
`rateIncreaseCriteria`		Os critérios para iniciar o aumento na taxa de implantação de um trabalho. AWS IoT suporta até um dígito após o 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 de configuração para pré-assinados URLs.
`expiresInSec`	longo	Por quanto tempo (em segundos) os pré-assinados URLs são válidos. Os valores válidos são 60 – 3600, o valor padrão é 1800 segundos. Os pré-assinados URLs são gerados quando uma solicitação do documento de trabalho é recebida.
`awsJobAbortConfig`		Os critérios que determinam quando e como ocorre a anulação de um trabalho.

Name	Tipo	Descrição
`abortCriteriaList`	list	A lista de critérios que determinam quando e como anular o trabalho.
`failureType`	string	O tipo de falhas de execução de trabalho que podem iniciar uma anulação. enum: FAILED | REJECTED | TIMED_OUT | ALL
`action`	string	O tipo de ação do trabalho a ser executada para iniciar a anulação. enum: CANCEL
`minNumberOfExecutedThings`	integer (mín:1)	O número mínimo de coisas que devem receber notificações de execução de trabalho antes que ele possa ser anulado.
`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` .

Name	Tipo	Descrição
`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.
`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.

Name	Tipo	Descrição
`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.
`version`	string	A versão do bucket do S3.
`codeSigning`		O método de assinatura do código do arquivo.
`awsSignerJobId`	string	O ID do AWSSigner Job 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.

Name	Tipo	Descrição
`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.
`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.

Name	Tipo	Descrição
`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 name/attribute pares.
`roleArn`	string (máx: 2048 mín:20)	A função do IAM que concede AWS IoT acesso ao Amazon S3, aos AWS IoT trabalhos e aos recursos de assinatur a de AWS código para criar um trabalho de atualização do OTA.
`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.

Name	Tipo	Descrição
`Value`	string (máx: 256 mín:1)	O valor da tag.

Saída

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

AWS CLI campos de saída

Name	Tipo	Descrição
`otaUpdateId`	string (máx: 128 mín:1)	O ID da atualização OTA
`awsIotJobId`	string	O ID do AWS IoT trabalho associado à atualização do OTA.
`otaUpdateArn`	string	O ARN da atualização OTA.
`awsIotJobArn`	string	O ARN do AWS IoT trabalho associado à atualização do 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 passado para o create-ota-update comando que usa a assinatura de código 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 passado para o create-ota-update AWS CLI comando que usa um arquivo embutido para fornecer material personalizado de assinatura de código.

```
[
  {
    "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 passado para o create-ota-update AWS CLI comando que permite que o FreeRTOS OTA inicie um trabalho de assinatura de código e crie um perfil e um stream de assinatura de código.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "myTestProfile",
        "signingProfileParameter": {
          "certificateArn": "your_certificate_arn",
          "platform": "your_platform_id",
          "certificatePathOnDevice": "certificate_path"
        },
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
    }
  }
]
```

Veja a seguir um exemplo de um arquivo JSON passado para o create-ota-update AWS CLI comando 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 passado para o create-ota-update AWS CLI comando que permite que o FreeRTOS OTA crie um stream 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 passado para o create-ota-update AWS CLI comando 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 list-ota-updates AWS CLI comando para obter uma lista de todas as atualizações do 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 get-ota-update AWS CLI comando para obter o status de criação ou exclusão de uma atualização do 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

Atualmente, você não pode usar o AWS IoT console para excluir streams ou atualizações OTA. Você pode usar o AWS CLI para excluir fluxos, atualizações de OTA e AWS IoT trabalhos criados durante uma atualização de OTA.

Exclusão de um fluxo do OTA

Ao criar uma atualização OTA que usa o MQTT, você pode usar a linha de comando ou o AWS IoT console para criar um fluxo para dividir o firmware em partes para que ele possa ser enviado pelo MQTT. Você pode excluir esse fluxo com o delete-stream AWS CLI comando, 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 AWS IoT trabalho para realizar a atualização.
- Uma execução de AWS IoT trabalho 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 AWS IoT trabalho associado à atualização do 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 AWS IoT um trabalho quando você cria uma atualização OTA. Uma execução de trabalho também é criada para cada dispositivo que processa o trabalho. Você pode usar o delete-job AWS CLI comando para excluir um trabalho e suas 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 a API do 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 no delete-job AWS CLI comando, você pode passar o `--force` parâmetro para o delete-job-execution para forçar a exclusão da execução de um trabalho. Para obter mais informações, consulte DeleteJobExecutionAPI.

> **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 a aplicação de demonstração de atualização OTA, consulte Over-the-air atualiza o aplicativo de demonstração.

Serviço do gerenciador de atualização OTA

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O serviço over-the-air (OTA) Update Manager fornece uma maneira de:

- Crie uma atualização OTA e os recursos que ela usa, incluindo um AWS IoT trabalho, um AWS IoT stream e assinatura de código.
- Obtenha informações sobre uma atualização OTA.
- Liste todas as atualizações do OTA associadas à sua AWS conta.
- 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.
- A função do IAM que concede AWS IoT acesso ao Amazon S3, aos AWS IoT trabalhos e aos recursos de assinatura de AWS código para criar um trabalho de atualização do 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 os arquivos que desejar para um ou mais dispositivos registrados 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 s atualizações de firmware com o Code Signing para FreeRTOS ou usar as próprias ferramentas de assinatura de código.

Para ter mais controle sobre o processo, você pode usar a CreateStreamAPI para criar um stream 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 o próprio agente OTA para detectar e fazer download das atualizações e realizar operações de instalação.

Integração do agente OTA à aplicação

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O agente over-the-air (OTA) foi projetado para simplificar a quantidade de código que você deve escrever para adicionar a funcionalidade de atualização OTA ao seu 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.

> **Note**
>
> Embora a integração do recurso de atualização OTA na aplicação seja bastante simples, o sistema de atualização OTA exige uma compreensão mais ampla do que apenas integração

de código de dispositivo. Para se familiarizar com a configuração de sua AWS conta com AWS IoT itens, credenciais, certificados de assinatura de código, dispositivos de provisionamento e tarefas 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 AWS IoT serviços, mas não gerencia a conexão MQTT. Para assegurar que o agente OTA não interfira na política de gerenciamento de conexão da aplicação, a conexão MQTT (incluindo a desconexão e qualquer funcionalidade de reconexão) deve ser processada pela aplicação 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 do arquivo. Se você escolher HTTP, o agente OTA manipulará as conexões HTTP.

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 conclusão 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 do OTA também mostra o gerenciamento da conexão MQTT ao monitorar 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 um jitter também é adicionada. Se a conexão for restabelecida, o agente OTA continuará suas operações.

Para ver um exemplo prático que usa o corretor AWS IoT MQTT, consulte o código de demonstração OTA no `demos/ota` diretório.

Como o agente OTA é sua própria tarefa, o atraso intencional de um segundo nesse exemplo afeta apenas essa aplicação. Não tem impacto no desempenho do agente.

```
void vRunOTAUpdateDemo( bool awsIotMqttMode,
                        const char * pIdentifier,
```

```
                        void * pNetworkServerInfo,
                        void * pNetworkCredentialInfo,
                        const IotNetworkInterface_t * pNetworkInterface )
{
    OTA_State_t eState;
    static OTA_ConnectionContext_t xOTAConnectionCtx;

    IotLogInfo( "OTA demo version %u.%u.%u\r\n",
                xAppFirmwareVersion.u.x.ucMajor,
                xAppFirmwareVersion.u.x.ucMinor,
                xAppFirmwareVersion.u.x.usBuild );

    for( ; ; )
    {
        IotLogInfo( "Connecting to broker...\r\n" );

        /* Establish a new MQTT connection. */
        if( _establishMqttConnection( awsIotMqttMode,
                                      pIdentifier,
                                      pNetworkServerInfo,
                                      pNetworkCredentialInfo,
                                      pNetworkInterface,
                                      &_mqttConnection ) == EXIT_SUCCESS )
        {
            /* Update the connection context shared with OTA Agent.*/
            xOTAConnectionCtx.pxNetworkInterface = ( void * ) pNetworkInterface;
            xOTAConnectionCtx.pvNetworkCredentials = pNetworkCredentialInfo;
            xOTAConnectionCtx.pvControlClient = _mqttConnection;

            /* Set the base interval for connection retry.*/
            _retryInterval = OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS;

            /* Update the connection available flag.*/
            _networkConnected = true;

            /* Check if OTA Agent is suspended and resume.*/
            if( ( eState = OTA_GetAgentState() ) == eOTA_AgentState_Suspended )
            {
                OTA_Resume( &xOTAConnectionCtx );
            }

            /* Initialize the OTA Agent , if it is resuming the OTA statistics will be
 cleared for new connection.*/
            OTA_AgentInit( ( void * ) ( &xOTAConnectionCtx ),
```

```
                            ( const uint8_t * ) ( clientcredentialIOT_THING_NAME ),
                            App_OTACompleteCallback,
                            ( TickType_t ) ~0 );

            while( ( ( eState = OTA_GetAgentState() ) != eOTA_AgentState_Stopped ) &&
_networkConnected )
            {
                /* Wait forever for OTA traffic but allow other tasks to run and output
statistics only once per second. */
                IotClock_SleepMs( OTA_DEMO_TASK_DELAY_SECONDS * 1000 );

                IotLogInfo( "State: %s  Received: %u   Queued: %u   Processed: %u
Dropped: %u\r\n", _pStateStr[ eState ],
                             OTA_GetPacketsReceived(), OTA_GetPacketsQueued(),
OTA_GetPacketsProcessed(), OTA_GetPacketsDropped() );
            }

            /* Check if we got network disconnect callback and suspend OTA Agent.*/
            if( _networkConnected == false )
            {
                /* Suspend OTA agent.*/
                if( OTA_Suspend() == kOTA_Err_None )
                {
                    while( ( eState = OTA_GetAgentState() ) !=
eOTA_AgentState_Suspended )
                    {
                        /* Wait for OTA Agent to process the suspend event. */
                        IotClock_SleepMs( OTA_DEMO_TASK_DELAY_SECONDS * 1000 );
                    }
                }
            }
            else
            {
                /* Try to close the MQTT connection. */
                if( _mqttConnection != NULL )
                {
                    IotMqtt_Disconnect( _mqttConnection, 0 );
                }
            }
        }
        else
        {
            IotLogError( "ERROR:  MQTT_AGENT_Connect() Failed.\r\n" );
        }
```

```
        /* After failure to connect or a disconnect, delay for retrying connection. */
        _connectionRetryDelay();
    }
}
```

Veja a seguir o fluxo de alto nível desta aplicação de demonstração:

- Crie um contexto do agente MQTT.
- Conecte-se ao endpoint da AWS IoT .
- Inicialize o agente OTA.
- Loop permitindo um trabalho de atualização OTA e estatísticas de saída uma vez por segundo.
- Se o MQTT se desconectar, suspenda as operações do agente OTA.
- Tente se conectar novamente com atraso e jitter exponencial.
- Se reconectado, retome as operações do agente OTA.
- Se o agente parar, espere um segundo e tente se reconectar.

Como usar um retorno de chamada personalizado para eventos de conclusão de OTA

O exemplo anterior usou `App_OTACompleteCallback` como manipulador de retorno de chamada para eventos de conclusão OTA. (Veja o terceiro parâmetro para a chamada de API `OTA_AgentInit`.) `App_OTACompleteCallback` é definido em *`freertos`*`/demos/ota/aws_iot_ota_update_demo.c`. Se você deseja implementar a manipulação personalizada dos eventos de conclusão, é necessário enviar o endereço da função do manipulador de retorno de chamada à API `OTA_AgentInit`. Durante o processo do OTA, o agente pode enviar um dos seguintes enums de eventos ao manipulador de retorno de chamada. Cabe ao desenvolvedor da aplicação decidir como e quando processar esses eventos.

```
/**
* @brief OTA Job callback events.
*
* After an OTA update image is received and authenticated, the Agent calls the user
* callback (set with the OTA_AgentInit API) with the value eOTA_JobEvent_Activate 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 eOTA_JobEvent_StartTest, 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 eOTA_JobEvent_Fail instead to allow the user to log the failure and take
* any action deemed appropriate by the user code.
*
*/
typedef enum {
    eOTA_JobEvent_Activate,  /*! OTA receive is authenticated and ready to activate. */
    eOTA_JobEvent_Fail,      /*! OTA receive failed. Unable to use this update. */
    eOTA_JobEvent_StartTest  /*! OTA job is now in self test, perform user tests. */
} OTA_JobEvent_t;
```

O agente OTA pode receber uma atualização em segundo plano durante o processamento ativo da aplicação principal. A finalidade de fornecer esses eventos é permitir que a aplicação decida se uma ação pode ser executada imediatamente ou se deve ser adiada até após a conclusão de algum outro processamento específico da aplicação. 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:

`eOTA_JobEvent_Activate event`

Quando esse evento é recebido pelo manipulador de retorno de chamada, 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.

`eOTA_JobEvent_Fail event`

A atualização falhou quando esse evento for recebido pelo manipulador de retorno de chamada. Não é necessário fazer nada nesse caso. Você pode querer produzir uma mensagem de log ou fazer algo específico para a aplicação.

`eOTA_JobEvent_StartTest event`

A fase de teste automático destina-se a permitir que o firmware recém-atualizado seja executado e testado antes de determinar se está funcionando corretamente e confirmá-lo como a última imagem permanente da aplicação. Quando uma nova atualização for recebida e autenticada e o dispositivo tiver sido redefinido, o agente OTA enviará o evento `eOTA_JobEvent_StartTest` 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( eOTA_ImageState_Accepted )`.

Se o dispositivo não tiver hardware ou mecanismos especiais que precisem ser testados, você poderá usar o manipulador de retorno de chamada padrão. Após o recebimento do evento `eOTA_JobEvent_Activate`, o manipulador padrão redefine o dispositivo imediatamente.

Segurança de OTA

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 do AWS IoT dispositivo e usa mecanismos AWS IoT de segurança. 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 FreeRTOS OTA Update Manager usa a Assinatura de Código AWS IoT para assinar automaticamente seu firmware. 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, você deve ter permissões para invocar o `CreateDeploymentCreateJob`, e. `CreateStream` APIs Além disso, em sua política de bucket ou ACL do Amazon S3, você deve conceder permissões de leitura ao responsável pelo AWS IoT serviço para que a atualização do firmware armazenada no Amazon S3 possa ser acessada durante o streaming.

Assinatura de código para AWS IoT

O AWS IoT console usa a Assinatura de Código AWS IoT para assinar automaticamente sua imagem de firmware para qualquer dispositivo compatível com AWS IoT.

A assinatura de código 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 (AC) 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 sua imagem de código, consulte o Guia de assinatura de código para AWS IoT desenvolvedores e a referência de assinatura de código para AWS IoT API.

> **Note**
>
> Você pode baixar o Code Signing for AWS IoT SDK em Tools for Amazon Web Services.

Solução de problemas do OTA

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

As seções a seguir contêm informações para ajudar você a solucionar problemas com atualizações OTA.

Tópicos

- Configurar CloudWatch registros para atualizações do OTA
- Registre chamadas da API AWS IoT OTA com AWS CloudTrail
- Obtenha Criar detalhes da OTAUpdate falha usando o AWS CLI
- Obtenha códigos de falha OTA com o AWS CLI
- Solução de problemas de atualizações de OTA de vários dispositivos
- Solucione problemas de atualizações OTA com o Texas Instruments CC322 0SF Launchpad

Configurar CloudWatch registros para atualizações do OTA

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O serviço de atualização OTA oferece suporte ao registro na Amazon CloudWatch. Você pode usar o AWS IoT console para habilitar e configurar o Amazon CloudWatch Logging 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 ativar o registro de atualizações do OTA, certifique-se de entender as permissões de acesso aos CloudWatch registros. Os usuários com acesso aos CloudWatch registros podem ver suas informações de depuração. Para obter informações, consulte Autenticação e controle de acesso para 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. Selecione 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 AWS IoT emprego é criado.
- Um AWS IoT trabalho está concluído.
- Um fluxo é criado.

Você pode ver os logs no console do CloudWatch .

Para ver uma atualização do OTA em CloudWatch Logs

1. Na página de navegação, escolha Logs.
2. Em Grupos de registros, escolha AWSIoTLogsV2.

Os logs de atualização OTA podem conter as seguintes propriedades:

accountId

O ID da AWS conta na qual o registro 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 AWS IoT emprego foi criado.
- `CreateStream`: um fluxo foi criado.
- `GetStream`: uma solicitação de stream foi enviada para o recurso de entrega de arquivos AWS IoT baseado em MQTT.
- `DescribeStream`: uma solicitação de informações sobre um fluxo foi enviada para o recurso de entrega de arquivos baseado em MQTT do AWS IoT .

awsJobId

O ID do AWS IoT trabalho que gerou o registro.

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.

protocolo

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 AWS IoT stream 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 registro gerado quando um AWS IoT trabalho 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 fluxo do de um recurso 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."
}
```

Registre chamadas da API AWS IoT OTA com AWS CloudTrail

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O FreeRTOS é integrado CloudTrail com um serviço que AWS IoT captura chamadas de API OTA e entrega os arquivos de log para um bucket do Amazon S3 que você especificar. CloudTrail captura chamadas de API do seu código para o AWS IoT OTA APIs. Usando as informações coletadas por CloudTrail, você pode determinar a solicitação que foi feita à AWS IoT OTA, 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 CloudTrail, inclusive como configurá-lo e ativá-lo, consulte o Guia AWS CloudTrail do usuário.

Informações sobre FreeRTOS em CloudTrail

Quando o CloudTrail registro está ativado em sua AWS conta, as chamadas de API feitas para ações AWS IoT OTA são rastreadas em arquivos de CloudTrail log, onde são gravadas com outros registros AWS de serviço. CloudTrail determina quando criar e gravar em um novo arquivo com base no período e no tamanho do arquivo.

As seguintes ações do plano de controle AWS IoT OTA são registradas por: CloudTrail

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream
- CriarOTAUpdate
- Obtenha OTAUpdate
- ListaOTAUpdates
- ExcluirOTAUpdate

> **Note**
>
> AWS IoT As ações do plano de dados OTA (lado do dispositivo) não são registradas por CloudTrail. Use 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 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 AWS serviço.

Para obter mais informações, consulte o elemento CloudTrail UserIdentity. AWS IoT As ações do OTA estão documentadas na Referência da API AWS IoT OTA.

É possível 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 criptografia do lado do servidor (SSE) do Amazon S3.

Se você quiser ser notificado quando os arquivos de log forem entregues, você pode configurar CloudTrail para publicar notificações do Amazon SNS. Para obter mais informações, consulte Configurando notificações do Amazon SNS para. CloudTrail

Você também pode agregar arquivos de log AWS IoT OTA de várias AWS regiões e várias AWS contas em um único bucket do Amazon S3.

Para obter mais informações, consulte Recebendo arquivos de CloudTrail log de várias regiões e Recebendo arquivos de CloudTrail log de várias contas.

Noções básicas das entradas de arquivo de log do FreeRTOS

CloudTrail os arquivos de log 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 CloudTrail registro que demonstra o registro de uma chamada à `CreateOTAUpdate` ação.

```
{
    "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"
}
```

Obtenha Criar detalhes da OTAUpdate falha usando o AWS CLI

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 de OTA, o serviço de gerenciamento de OTA cria um trabalho de IoT e o programa para os dispositivos de destino, e esse processo também cria ou usa outros tipos de AWS recursos em sua conta (um trabalho de assinatura de código, um stream, AWS IoT um objeto do Amazon S3). Qualquer erro encontrado pode fazer com que o processo falhe sem criar um AWS IoT trabalho. 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
   ```

   Onde *ota_update_job_001* está o ID que você deu à atualização do OTA quando a criou.
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.

Obtenha códigos de falha OTA com o AWS CLI

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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
```

Onde *JobID* está a string completa de ID do trabalho cujo status queremos obter (estava associado ao trabalho de atualização do OTA quando foi criado) e *ThingName* é o nome do AWS IoT item com o qual o dispositivo está registrado em 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Para executar OTAs em vários dispositivos (coisas) usando a mesma imagem de firmware, implemente uma função (por exemplo`getThingName()`) que recupere `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.

Solucione problemas de atualizações OTA com o Texas Instruments CC322 0SF Launchpad

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

A plataforma CC322 0SF 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 restore-to-factory processo 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário 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 aplicações incorporadas. As bibliotecas do FreeRTOS também permitem que seus aplicativos interajam com os serviços. AWS IoT O FreeRTOS inclui bibliotecas que permitem:

- Conecte dispositivos com segurança à AWS IoT nuvem usando o MQTT e as sombras do dispositivo.
- Descubra e conecte-se aos AWS IoT Greengrass núcleos.
- Gerencie conexões de Wi-Fi.
- Ouça e processe Atualizações do FreeRTOS Over-the-Air.

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

Biblioteca	Referência da API	Descrição
Bluetooth Low Energy	Referência de API do Bluetooth Low Energy	Usando a biblioteca FreeRTOS Bluetooth Low Energy, seu microcontrolador pode se comunicar com AWS IoT o agente MQTT por meio de um dispositivo de gateway. Para obter mais informações, consulte Biblioteca de Bluetooth Low Energy.
Over-the-Air Atualizações	Referência de API do OTA Agent	A biblioteca do AWS IoT Over-the-Air FreeRTOS (OTA) Agent conecta seu dispositivo FreeRTOS ao Agente OTA. AWS IoT Para obter mais informações, consulte Biblioteca do agente OTA.
FreeRTOS+POSIX	Referência de API do FreeRTOS+POSIX	É possível usar a biblioteca do FreeRTOS+POSIX para transferir aplicações 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. Para obter mais informações, consulte FreeRTOS+TCP.

Biblioteca	Referência da API	Descrição
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.
núcleo PKCS11		A PKCS11 biblioteca principal é uma implementação de referênci a do Public Key Cryptography Standard #11, para oferecer suporte ao provisionamento e à autentica ção do cliente TLS. Para obter mais informações, consulte PKCS11 biblioteca principal.
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.

Bibliotecas de aplicações do FreeRTOS

Opcionalmente, você pode incluir as seguintes bibliotecas de aplicativos independentes em sua configuração do FreeRTOS para interagir com serviços na nuvem. AWS IoT

> **Note**
>
> Algumas das bibliotecas de aplicativos têm APIs as mesmas bibliotecas do AWS IoT Device SDK for Embedded C. Para essas bibliotecas, consulte a Referência da API C do AWS IoT Device SDK. Para obter mais informações sobre o AWS IoT Device SDK for Embedded C, consulteAWS IoT SDK de dispositivo para C incorporado.

Bibliotecas de aplicações do FreeRTOS

Biblioteca	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 FreeRTOS conecta seu dispositivo FreeRTOS a. AWS IoT Device Defender Para obter mais informações, consulte AWS IoT Device Defender biblioteca.
AWS IoT Greengrass	Referência de API do Greengrass	A biblioteca AWS IoT Greengrass FreeRTOS conecta seu dispositivo FreeRTOS a. AWS IoT Greengrass Para obter mais informações, consulte AWS IoT Greengrass Biblioteca Discovery.
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. O MQTT é o protocolo que os dispositivos usam para interagir AWS IoT. Para obter mais informações sobre a versão 3.0.0 da bibliotec a coreMQTT, consulte Biblioteca coreMQTT.
AWS IoT Device Shadow	Referência de API para o SDK de C do device shadow	A biblioteca AWS IoT Device Shadow permite que seu dispositivo FreeRTOS interaja AWS IoT com as sombras do dispositivo.

Biblioteca	Referência da API	Descrição
		Para obter mais informações, consulte AWS IoT Biblioteca Device Shadow.

Bibliotecas comuns do FreeRTOS

As seguintes bibliotecas comuns estendem a funcionalidade do kernel com estruturas de dados adicionais e funções para desenvolvimento incorporado de aplicações. Essas bibliotecas são frequentemente dependências da portabilidade e das bibliotecas de aplicações do FreeRTOS.

> **Note**
>
> O AWS IoT Device SDK for Embedded C inclui bibliotecas comuns com APIs funcionalidades idênticas a essas bibliotecas. Para obter uma referência de API, consulte a Referência de API em C do AWS IoT Device SDK.

Bibliotecas comuns do FreeRTOS

Biblioteca	Referência da API	Descrição
Operações atômicas	Referência de API para o SDK de C de operações atômicas	Para obter mais informações, consulte Operações atômicas.
Contêineres lineares	Referência de API para SDK de C de contêineres lineares	Para obter mais informações, consulte Biblioteca Linear Containers.
Registro em log	Referência de API para o SDK de C de registro	Para obter mais informações, consulte Biblioteca de registros em log.
Memória estática	Referência de API para o SDK de C de memória estática	Para obter mais informações, consulte Biblioteca de memória estática.

Biblioteca	Referência da API	Descrição
Grupo de tarefas	Referência de API para o SDK de C de grupo de tarefas	Para obter mais informações, consulte Biblioteca de Grupo de tarefas.

Configuração das bibliotecas do FreeRTOS

As configurações do FreeRTOS e AWS IoT do Device SDK for Embedded C 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.

Bibliotecas comuns

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O FreeRTOS inclui algumas bibliotecas comuns que ampliam a funcionalidade do kernel com estruturas de dados adicionais e funções para o desenvolvimento incorporado de aplicações. Essas bibliotecas são frequentemente dependências de outras bibliotecas do FreeRTOS.

Tópicos

- Operações atômicas
- Biblioteca Linear Containers

- Biblioteca de registros em log
- Biblioteca de memória estática
- Biblioteca de Grupo de tarefas

Operações atômicas

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Visão geral

Operações atômicas garantem uma sincronização sem bloqueios na programação simultânea. Você pode usar operações atômicas para resolver problemas de desempenho causados por operações assíncronas que atuam em memória compartilhada. Há suporte para operações atômicas no FreeRTOS, conforme implementado no arquivo de cabeçalho `iot_atomic.h`.

O arquivo de cabeçalho `iot_atomic.h` inclui duas implementações para operações atômicas:

- Desativar a interrupção global.

  Essa implementação está disponível para todas as plataformas do FreeRTOS.
- Suporte nativo ISA atômico.

  Essa implementação está disponível apenas para plataformas que compilam com GCC, versão 4.7.0 e superior, e têm suporte atômico ISA. Para obter informações sobre funções integradas GCC, consulte Built-in Functions for Memory Model Aware Atomic Operations.

Inicialização

Antes de usar as operações atômicas do FreeRTOS, é necessário escolher qual implementação de operações atômicas você deseja usar.

1. Abra o arquivo de configuração `FreeRTOSConfig.h` para editar.

2. Para a implementação de suporte atômico nativo ISA, defina e configure a variável `configUSE_ATOMIC_INSTRUCTION` como 1.

   Para a implementação de desativação da interrupção global, cancele ou limpe a definição `configUSE_ATOMIC_INSTRUCTION`.

Referência da API

Para obter uma referência completa de API, consulte Referência de API para o SDK de C de operações atômicas.

Biblioteca Linear Containers

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

A biblioteca de contêineres lineares do FreeRTOS define estruturas de dados lineares, incluindo listas e filas, para você utilizar no desenvolvimento de aplicações incorporadas.

Referência da API

Para obter uma referência completa de API, consulte Referência de API para SDK de C de contêineres lineares.

Biblioteca de registros em log

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

A biblioteca de registro em log do FreeRTOS permite que as bibliotecas do FreeRTOS gravem mensagens no log de depuração.

Referência da API

Para obter uma referência completa de API, consulte Referência de C SDKAPI de registro em log.

Biblioteca de memória estática

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

A biblioteca de memória estática do FreeRTOS define algumas funções para o gerenciamento de buffers estáticos. Nessa biblioteca, você pode usar um componente de memória estática para fornecer buffers alocados estaticamente, em vez de usar a alocação de memória dinâmica.

Referência da API

Para obter uma referência completa de API, consulte Referência de API para o SDK de C de memória estática.

Biblioteca de Grupo de tarefas

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Visão geral

O FreeRTOS é compatível com o gerenciamento de tarefas usando a biblioteca de Pool de tarefas do FreeRTOS. A biblioteca do Grupo de tarefas permite agendar tarefas em segundo plano, além de agendar e cancelar tarefas assíncronas com segurança. Usando o Pool de Tarefas APIs, você pode configurar as tarefas do seu aplicativo para otimizar a compensação entre desempenho e espaço ocupado pela memória.

A biblioteca do Grupo de tarefas é feita de duas estruturas de dados principais: o Grupo de tarefas e os trabalhos do Grupo de tarefas.

Grupo de tarefas (**`IotTaskPool_t`**)

O Grupo de tarefas contém uma fila de expedição que gerencia a fila de trabalhos a serem executados, além de gerenciar os threads de operadores que executam os trabalhos.

Trabalhos de grupo de tarefas (**`IotTaskPoolJob_t`**)

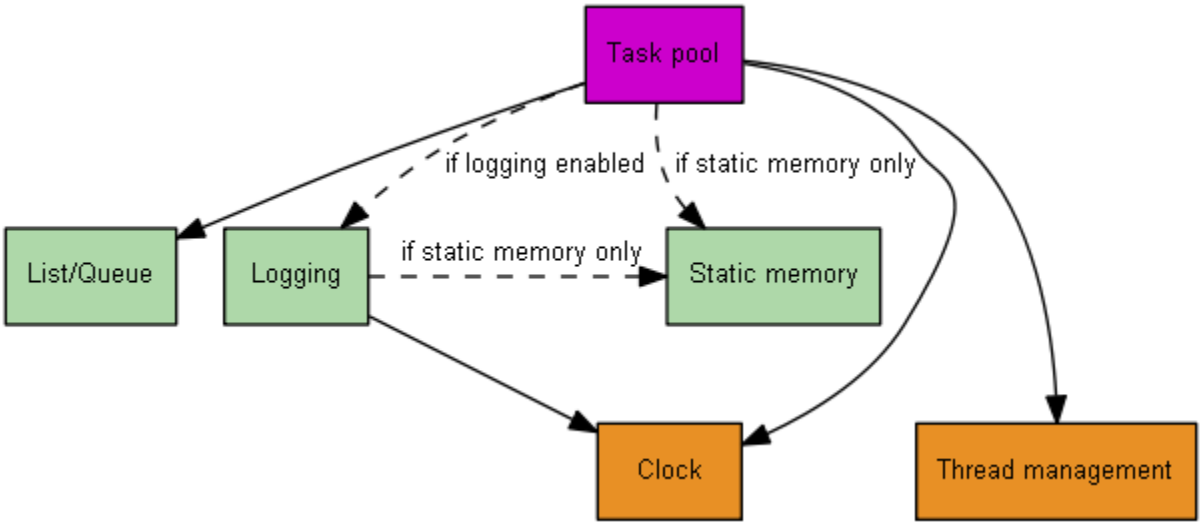
Os trabalhos do Grupo de tarefas podem ser executados como trabalhos em segundo plano ou trabalhos temporizados em segundo plano. Os trabalhos em segundo plano são iniciados em First-In-First-Out ordem e não têm restrições de tempo. Os trabalhos temporizados são programados para execução em segundo plano e seguem um temporizador.

> **Note**
>
> O grupo de tarefas pode garantir apenas que um trabalho temporizado será executado após um determinado momento, e não dentro de um intervalo de tempo específico.

Dependências e requisitos

A biblioteca do Grupo de tarefas tem as seguintes dependências:

- A biblioteca de contêineres lineares (lista/fila) para manter as estruturas de dados de operações de grupo de tarefas programadas e em andamento.
- A biblioteca de registro em log (se a configuração `IOT_LOG_LEVEL_TASKPOOL` não for `IOT_LOG_NONE`).
- Uma camada de plataforma que fornece uma interface para o sistema operacional para o gerenciamento de threads, temporizadores, funções de relógio, etc.



Atributos

Usando a biblioteca do Pool de Tarefas APIs, você pode fazer o seguinte:

- Programar trabalhos imediatos e adiados com as funções da API sem bloqueio da biblioteca.
- Criar trabalhos alocados estaticamente e dinamicamente.
- Configurar a biblioteca para escalar o desempenho e a capacidade, de acordo com os recursos de seu sistema.
- Personalizar o armazenamento em cache para a sobrecarga de memória baixa ao criar trabalhos dinamicamente.

Solução de problemas

As funções da biblioteca de Grupo de tarefas retornam códigos de erro como valores enumerados `IotTaskPoolError_t`. Para obter mais informações sobre cada código de erro, consulte a documentação de referência dos tipos de dados enumerados `IotTaskPoolError_t` em Referência de API para o SDK de C de grupo de tarefas.

Restrições de uso

A biblioteca de Grupo de tarefas não pode ser usada em uma rotina de serviço de interrupção (ISR).

Não recomendamos retornos de chamada de usuários do grupo de tarefas que realizam operações de bloqueio, especialmente operações de bloqueio indefinidas. As operações de bloqueio de longa duração basicamente roubam um thread do grupo de tarefas e criam um possível bloqueio ou esgotamento.

Inicialização

Antes de usar o grupo de tarefas, uma aplicação precisa chamar `IotTaskPool_CreateSystemTaskPool` para iniciar uma instância de um grupo de tarefas do sistema. A aplicação precisa garantir que o grupo de tarefas no nível do sistema seja inicializado com antecedência suficiente na sequência de inicialização, antes de qualquer biblioteca usá-lo e antes de qualquer código de aplicação publicar um trabalho no grupo de tarefas. Após a inicialização, o sistema inicia o grupo de tarefas único no sistema compartilhado por todas as bibliotecas. Após a inicialização, o identificador do grupo de tarefas pode ser recuperado para utilização com a API `IOT_SYSTEM_TASKPOOL`.

> **Note**
>
> Chamar `IotTaskPool_CreateSystemTaskPool` não aloca memória para armazenar as estruturas de dados e os estados do grupo de tarefas, no entanto, pode alocar memória para armazenar as entidades dependentes e as estruturas de dados, como por exemplo, os threads do grupo de tarefas.

Referência da API

Para obter uma referência completa de API, consulte Referência de API para o SDK de C de grupo de tarefas.

Exemplo de uso

Suponha que você precise agendar uma coleção recorrente de AWS IoT Device Defender métricas e decida usar um cronômetro para agendar a coleta com chamadas para a conexão, assinatura e publicação do MQTT. APIs O código a seguir define uma função de retorno de chamada para aceitar AWS IoT Device Defender métricas em todo o MQTT, com um retorno de chamada de desconexão que se desconecta da conexão MQTT.

```
/* An example of a user context to pass to a callback through a task pool thread. */
typedef struct JobUserContext
{
    uint32_t counter;
} JobUserContext_t;

/* An example of a user callback to invoke through a task pool thread. */
```

```
static void ExecutionCb( IotTaskPool_t * pTaskPool, IotTaskPoolJob_t * pJob, void *
 context )
{
    ( void )pTaskPool;
    ( void )pJob;
    JobUserContext_t * pUserContext = ( JobUserContext_t * )context;
    pUserContext->counter++;
}

void TaskPoolExample( )
{
    JobUserContext_t userContext = { 0 };
    IotTaskPoolJob_t job;
    IotTaskPool_t * pTaskPool;
    IotTaskPoolError_t errorSchedule;

    /* Configure the task pool to hold at least two threads and three at the maximum.
 */
    /* Provide proper stack size and priority per the application needs. */
    const IotTaskPoolInfo_t tpInfo = { .minThreads = 2, .maxThreads = 3, .stackSize =
 512, .priority = 0 };

    /* Create a task pool. */
    IotTaskPool_Create( &tpInfo, &pTaskPool );

    /* Statically allocate one job, then schedule it. */
    IotTaskPool_CreateJob( &ExecutionCb, &userContext, &job );
    errorSchedule = IotTaskPool_Schedule( pTaskPool, &job, 0 );

    switch ( errorSchedule )
    {
    case IOT_TASKPOOL_SUCCESS:
        break;
    case IOT_TASKPOOL_BAD_PARAMETER:          // Invalid parameters, such as a NULL
 handle, can trigger this error.
    case IOT_TASKPOOL_ILLEGAL_OPERATION:      // Scheduling a job that was previously
 scheduled or destroyed could trigger this error.
    case IOT_TASKPOOL_NO_MEMORY:              // Scheduling a with flag
 #IOT_TASKPOOL_JOB_HIGH_PRIORITY could trigger this error.
    case IOT_TASKPOOL_SHUTDOWN_IN_PROGRESS:   // Scheduling a job after trying to
 destroy the task pool could trigger this error.
        // ASSERT
        break;
    default:
```

```
        // ASSERT*/
    }

    /* ................................ */
    /*  ... Perform other operations ... */
    /* ................................ */

    IotTaskPool_Destroy( pTaskPool );
}
```

Biblioteca backoffAlgorithm

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 recuo exponencial com jitter algoritmo.

O recuo exponencial com jitter 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.

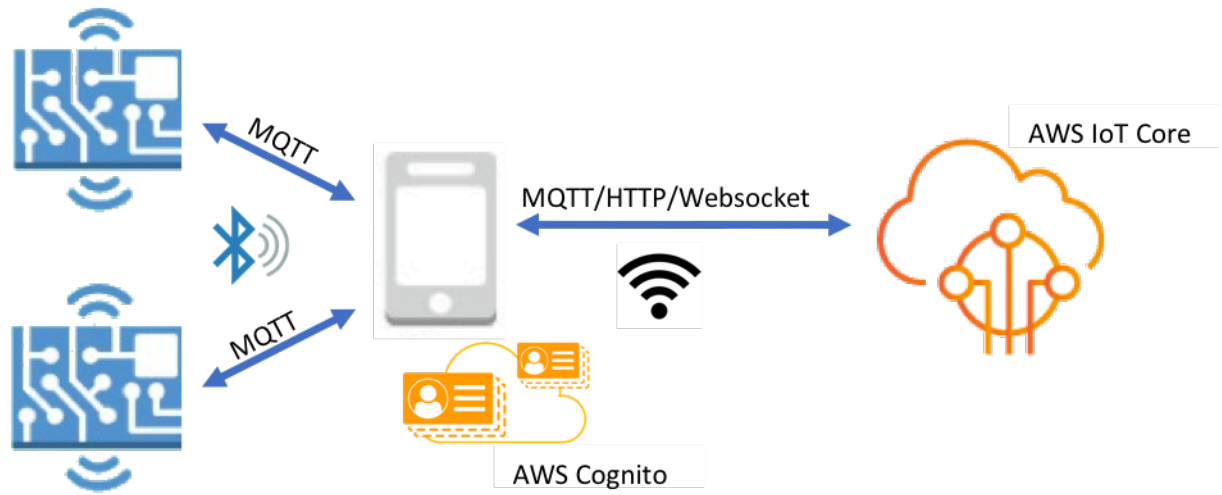
Tamanho de código de backoffAlgorithm (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
backoff_algorithm.c	0,1 K	0,1 K
Estimativas totais	0,1 K	0,1 K

Biblioteca de Bluetooth Low Energy

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Visão geral

O FreeRTOS oferece suporte à publicação e inscrição em tópicos do MQTT pela Bluetooth Low Energy por meio de um dispositivo de proxy, como um celular. Com a biblioteca FreeRTOS Bluetooth Low Energy, seu microcontrolador pode se comunicar com segurança com o broker MQTT. AWS IoT

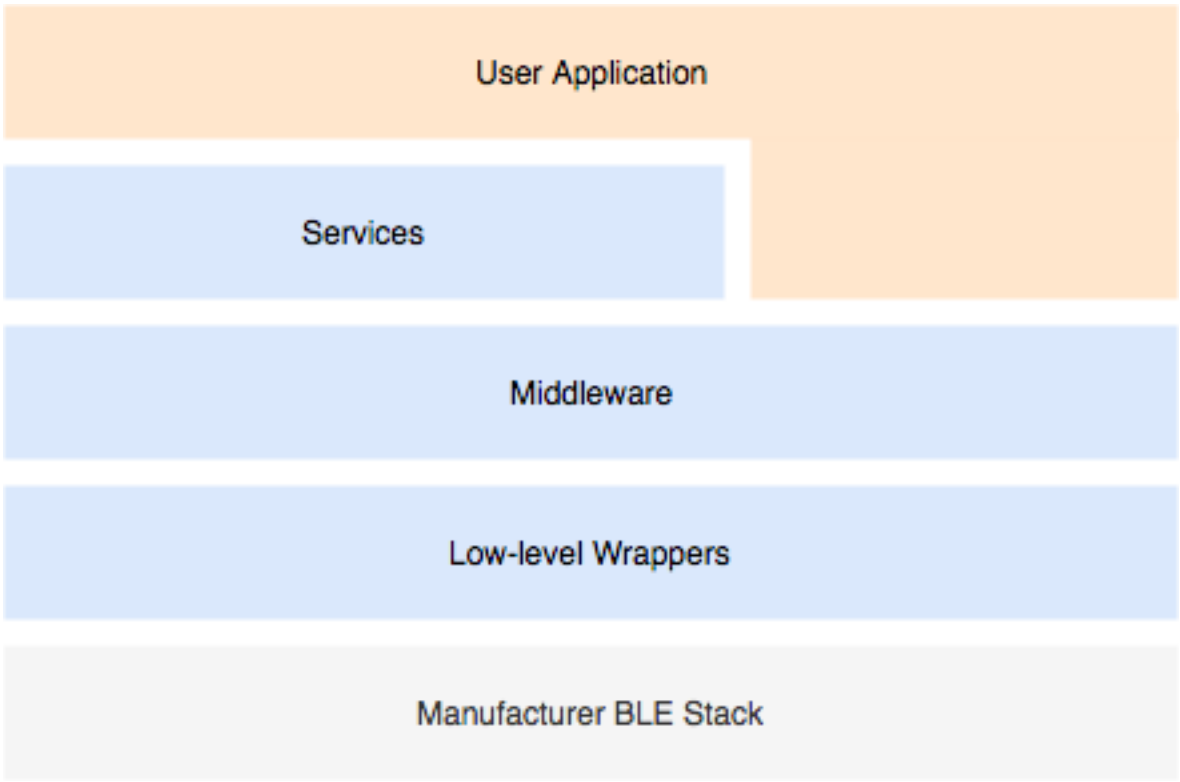


Usando o Mobile SDKs for FreeRTOS Bluetooth, você pode escrever aplicativos móveis nativos que se comunicam com os aplicativos incorporados em seu microcontrolador via Bluetooth Low Energy. Para obter mais informações sobre o celular SDKs, consulteCelular SDKs para dispositivos Bluetooth FreeRTOS.

A biblioteca Bluetooth Low Energy do FreeRTOS inclui serviços para configurar redes Wi-Fi, transferir grandes quantidades de dados e fornecer abstrações de rede pela Bluetooth Low Energy. A biblioteca FreeRTOS Bluetooth Low Energy também inclui middleware e APIs nível inferior para um controle mais direto sobre sua pilha Bluetooth Low Energy.

Arquitetura

Três camadas compõem a biblioteca Bluetooth Low Energy do FreeRTOS serviços, middleware e wrappers de nível inferior.



Serviços

A camada de serviços Bluetooth Low Energy do FreeRTOS consiste em quatro serviços de atributos genéricos (GATT) que utilizam o APIs middleware: informações do dispositivo, provisionamento de Wi-Fi, abstração de rede e transferência de objetos grandes.

Informações do dispositivo

O serviço de Informações do dispositivo reúne informações sobre o microcontrolador, incluindo:

- A versão do FreeRTOS que o 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 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.

Middleware

O middleware FreeRTOS Bluetooth Low Energy é uma abstração do nível inferior. APIs O middleware APIs compõe uma interface mais amigável para a pilha Bluetooth Low Energy.

Usando o middleware APIs, você pode registrar vários retornos de chamada, em várias camadas, em 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

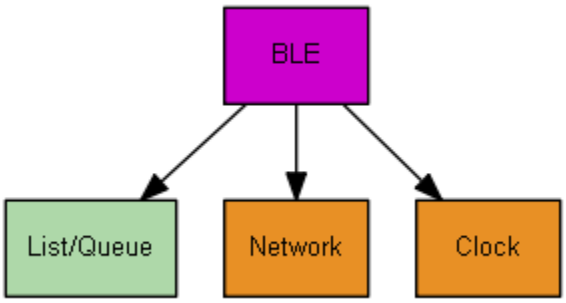
Os wrappers de nível baixo da Bluetooth Low Energy do FreeRTOS são uma abstração da pilha do fabricante Bluetooth Low Energy. Os invólucros de baixo nível oferecem um conjunto comum APIs de controle direto sobre o hardware. Os de baixo nível APIs otimizam o uso da RAM, mas têm funcionalidade limitada.

Use o serviço Bluetooth Low Energy APIs para interagir com os serviços Bluetooth Low Energy. O serviço APIs exige mais recursos do que o de baixo nível APIs.

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:

- O nível baixo APIs usa menos RAM, mas oferece funcionalidade limitada.
- 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`.

Nível baixo APIs

Se você não quiser usar os serviços FreeRTOS Bluetooth Low Energy GATT, você pode ignorar o middleware e interagir diretamente com os de baixo nível para economizar recursos. APIs

Para inicializar o nível baixo APIs

1. Inicialize todos os drivers de hardware Bluetooth Low Energy antes de ligar para o. APIs A inicialização do driver não faz parte do nível baixo APIs do 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 ligar para o APIs, você deve ativar o Bluetooth Low Energy.

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. O gerenciador de Bluetooth contém o APIs que é comum ao Bluetooth Low Energy e ao Bluetooth classic. 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 da 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 as características e os descritores do serviço:

```
#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 portar os FreeRTOS para um novo destino, você deve implementar APIs alguns para o serviço de provisionamento de Wi-Fi e a funcionalidade Bluetooth Low Energy.

Bluetooth de baixa energia APIs

Para usar o middleware FreeRTOS Bluetooth Low Energy, você deve implementar alguns. APIs

APIs comum 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 específico 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.

Celular SDKs para dispositivos Bluetooth FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Você pode usar o Mobile SDKs for FreeRTOS Bluetooth Devices para criar aplicativos móveis que interajam com seu microcontrolador via Bluetooth Low Energy. O Mobile também SDKs pode 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 o 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.

E/S comum

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 aplicação 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 o mesmo APIs para acessar vários drivers de dispositivos em várias placas de referência baseadas em microcontroladores (MCU).

A E/S comum do FreeRTOS atua como essa camada de abstração de hardware. Ele fornece um conjunto de padrões APIs para acessar dispositivos seriais comuns em placas de referência suportadas. Esses comuns APIs se comunicam e interagem com esses periféricos e permitem que seu código funcione em todas as 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.

AWS IoT Device Defender biblioteca

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

Você pode usar a AWS IoT Device Defender biblioteca para enviar métricas de segurança de seus 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 não parecer certo, AWS IoT Device Defender envia um alerta para que você possa tomar medidas para corrigir o problema. As interações com 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 AWS IoT Device Defender biblioteca pode ser usada livremente e é distribuída sob a licença de código aberto do MIT.

Tamanho do código do AWS IoT Device Defender (exemplo gerado com o 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

AWS IoT Greengrass Biblioteca Discovery

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 o AWS IoT Greengrass Discovery APIs, seu dispositivo pode enviar mensagens para um núcleo do Greengrass depois de encontrar o endpoint do núcleo.

Dependências e requisitos

Para usar a biblioteca do Greengrass Discovery, você deve criar algo 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`

Seu AWS IoT endpoint.

`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 Introdução ao 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 da 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 AWS IoT um arquivo JSON que contém os principais parâmetros de conectividade 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 alocar certificados dinamicamente, você deve chamar o seguinte: APIs

`GGD_JSONRequestStart`

Faz uma solicitação HTTP GET AWS IoT para iniciar a solicitação de descoberta para descobrir um núcleo do Greengrass. `GD_SecureConnect_Send`é usado para enviar a solicitação para 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 as aplicações possam gerenciar completamente sua simultaneidade. Ela usa somente buffers fixos, para que as aplicações tenham controle total de sua estratégia de alocação de memória.

A biblioteca é escrita em C e projetada 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 aplicações do IoT, recomendamos usar uma interface de transporte segura, como a usada pelo protocolo TLS, conforme demonstrado na demonstração coreHTTP (autenticação mútua).

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
http_parser.c (analisador http)	15,7 K	13,0 K
Estimativas totais	18,9 K	15,6 K

Biblioteca coreJSON

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

JSON (JavaScript Object Notation) é um formato de serialização de dados legível por humanos. É amplamente usado para trocar dados, como com o serviço AWS IoT Device Shadow, e faz parte de muitos APIs, como a API GitHub REST. 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

A biblioteca coreMQTT é uma implementação de cliente do padrão MQTT (Message Queue Telemetry Transport). O padrão MQTT fornece um protocolo leve de publicação/assinatura (ou PubSub) de mensagens que roda sobre TCP/IP e é frequentemente usado em casos de uso de máquina a máquina (M2M) e 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 (protocolo TLS). 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 da aplicação pode selecionar uma interface de transporte existente ou implementar a própria interface, conforme adequado para a sua aplicação.

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 que a serializer/deserializer API. This API can be used to build a simple IoT application consisting of only the required a subset of MQTT functionality, without any other overhead. The serializer/deserializer API de baixo nível pode ser usada em conjunto com qualquer API de camada de transporte disponível, como soquetes, para enviar e receber mensagens de e para o agente.

Ao usar conexões MQTT em aplicações 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 demonstram 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	3,0 K	2,6 K
core_mqtt_state.c	1,4 K	1,1 K
core_mqtt_serializer.c	2,5 K	2,0 K
Estimativas totais	6,9 K	5,7 K

Biblioteca do agente OTA

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Visão geral

A Biblioteca do agente OTA permite gerenciar as notificações, o download e a verificação de atualizações de firmware para dispositivos FreeRTOS usando HTTP ou MQTT como protocolo. Usando a biblioteca do agente OTA, você pode separar logicamente as atualizações de firmware e a aplicação em execução nos dispositivos. O agente OTA pode compartilhar uma conexão de rede com a aplicação. 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 da aplicação para testar, confirmar ou reverter uma atualização de firmware.

Para obter mais informações sobre como configurar atualizações OTA com o FreeRTOS consulte Atualizações do FreeRTOS Over-the-Air.

Atributos

Esta é a interface completa do agente OTA:

`OTA_AgentInit`

Inicializa o agente OTA. O chamador fornece o contexto do protocolo de mensagens, um retorno de chamada opcional e um tempo limite.

`OTA_AgentShutdown`

Remove os recursos depois de usar o agente OTA.

`OTA_GetAgentState`

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 agente OTA.

Referência da API

Para obter mais informações, consulte a Referência da API do agente OTA.

Exemplo de uso

Uma aplicação 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 corretor AWS IoT MQTT. Para obter mais informações, consulte Biblioteca coreMQTT.
2. Inicialização do agente OTA chamando `OTA_AgentInit`. A aplicação pode definir uma função de retorno de chamada OTA personalizada ou usar o retorno de chamada padrão, especificando um ponteiro de função de retorno de chamada NULL. Também é necessário fornecer um tempo limite de inicialização.

O retorno de chamada implementa a lógica específica da aplicação que é executada após a conclusão de um trabalho de atualização OTA. O tempo limite define quanto tempo aguardar a conclusão da inicialização.

3. Se o `OTA_AgentInit` expirou antes que o agente estivesse pronto, você pode chamar `OTA_GetAgentState` para confirmar que o agente foi inicializado e está operando conforme o esperado.
4. 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`.
5. Se a nova imagem de firmware tiver sido rejeitada (por exemplo, devido a um erro de validação), a aplicação normalmente poderá ignorar a notificação e aguardar a próxima atualização.
6. 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 do código do AWS IoT OTA (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização -Os
ota.c	8,0 K	7,2 K
ota_interface.c	0,1 mil	0,1 mil
ota_base64.c	0,6 mil	0,6 mil
ota_mqtt.c	2,3 K	2,2 K
ota_cbor.c	0,8 mil	0,6 mil
ota_http.c	0,3 mil	0,3 mil
Estimativas totais	12,1 K	11,0 K

PKCS11 biblioteca principal

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 da aplicação use, crie, modifique e exclua objetos criptográficos, sem expor esses objetos à memória da aplicação. Por exemplo, as integrações de referência do AWS FreeRTOS 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 nunca “veja” a chave.

A PKCS11 biblioteca principal contém uma implementação simulada baseada em software da interface PKCS #11 (API) que usa a funcionalidade criptográfica fornecida pelo Mbed TLS. 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 de PKCS11 software principal é, portanto, fornecer uma implementação PKCS #11 não específica de hardware que permita prototipagem e desenvolvimento rápidos antes de mudar para uma implementação PKCS #11 específica de 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.

Atributos

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 AWS IoT algo com base em um certificado de cliente P-256.

Verifique se você está usando as seguintes versões (ou mais recentes) do 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 .

Para criar AWS IoT algo com base em um certificado de cliente P-256

1. Crie qualquer AWS IoT coisa.

```
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 do certificado para 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, as aplicações devem armazenar esses arquivos em um local seguro.

Portabilidade

Para obter informações sobre como portar a PKCS11 biblioteca principal para sua plataforma, consulte Portando a PKCS11 biblioteca principal no Guia de portabilidade do FreeRTOS.

Uso de memória

Tamanho do código do núcleo PKCS11 (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização: Os
core_pkcs11.c	0,8 K	0,8 K

Tamanho do código do núcleo PKCS11 (exemplo gerado com GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização: Os
core_pki_utils.c	0,5 K	0,3 K
core_pkcs11_mbedtls.c	8,7 K	7,3 K
Estimativas totais	10,0 K	8,4 K

Biblioteca de Secure Sockets

Important

Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Visão geral

Use a biblioteca de Secure Sockets do FreeRTOS para criar aplicações incorporadas que se comunicam de maneira segura. A biblioteca foi projetada para facilitar a integração de desenvolvedores de software de várias bases de programação de rede.

A biblioteca de Secure Sockets do FreeRTOS se baseia na interface de soquetes Berkeley, com uma opção de comunicação segura adicional por protocolo TLS. Para obter informações sobre as diferenças entre a biblioteca de Secure Sockets do FreeRTOS e a interface de soquetes Berkeley, consulte `SOCKETS_SetSockOpt` na Referência de API de Secure Sockets.

Note

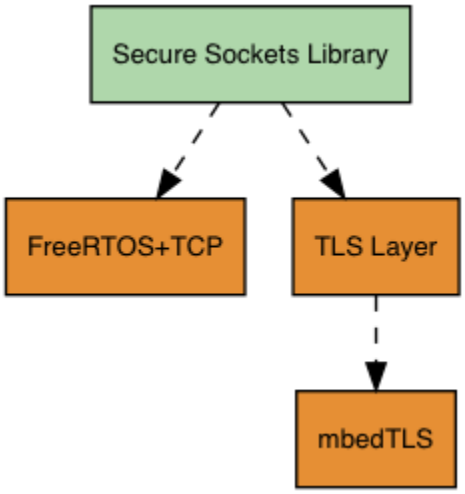
Atualmente, somente o cliente APIs, além de uma implementação IP leve (LWIP) da API do `Bind` lado do servidor, são compatíveis com os Freertos Secure Sockets.

Dependências e requisitos

A biblioteca de Secure Sockets do FreeRTOS depende de uma pilha TCP/IP e de uma implementação do TLS. As portas para o FreeRTOS atendem a essas dependências em uma de três maneiras:

- Uma implementação personalizada de TCP/IP e TLS
- Uma implementação personalizada de TCP/IP, e a camada TLS do FreeRTOS com mbedTLS
- FreeRTOS+TCP e a camada TLS do FreeRTOS com mbedTLS

O diagrama de dependências abaixo mostra a implementação de referência inclusa com a biblioteca de Secure Sockets do FreeRTOS. Essa implementação de referência oferece suporte a TLS e TCP/IP sobre Ethernet e Wi-Fi com FreeRTOS+TCP e mbedTLS como dependências. Para obter mais informações sobre a camada TLS do FreeRTOS, consulte Transport Layer Security.



Atributos

Os recursos da biblioteca de Secure Sockets do FreeRTOS incluem:

- Uma interface padrão baseada em soquetes Berkeley
- Thread-safe APIs para enviar e receber dados
- Easy-to-enable TLS

Solução de problemas

Códigos de erro

Os códigos de erro que a biblioteca de Secure Sockets do FreeRTOS retorna são valores negativos. Para obter mais informações sobre cada código de erro, consulte Códigos de erro de Secure Sockets na Referência da API de Secure Sockets.

> **Note**
>
> Se a API de Secure Sockets do FreeRTOS retornar um código de erro, o Biblioteca coreMQTT, que depende da biblioteca de Secure Sockets do FreeRTOS, retorna o código de erro `AWS_IOT_MQTT_SEND_ERROR`.

Suporte ao desenvolvedor

A biblioteca de Secure Sockets do FreeRTOS inclui duas macros auxiliares para manuseio de 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 soquetes TCP são compatíveis com a biblioteca de Secure Sockets do FreeRTOS. Os soquetes UDP não são compatíveis.

APIs Os servidores não são suportados pela biblioteca FreeRTOS Secure Sockets, exceto por uma implementação de IP leve (LWIP) da API do lado do servidor. `Bind` O cliente APIs é suportado.

Inicialização

Para usar a biblioteca de Secure Sockets do FreeRTOS, é necessário 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 FreeRTOS +TCP for uma dependência, também é necessário invocar `FreeRTOS_IPInit` na aplicação.

Referência da API

Para obter uma referência completa de API, consulte Referência de API do Secure Sockets.

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 Secure Sockets do FreeRTOS depende de uma pilha TCP/IP e de uma implementação do TLS. Dependendo da pilha, para transferir a biblioteca de Secure Sockets, será necessário transferir alguns dos seguintes:

- A pilha TCP/IP do FreeRTOS+TCP
- A PKCS11 biblioteca principal
- A Transport Layer Security

Para obter mais informações sobre portabilidade, consulte Portabilidade da biblioteca de Secure Sockets no Guia de portabilidade do FreeRTOS.

AWS IoT Biblioteca Device Shadow

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

Você pode usar a biblioteca 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 suas aplicações 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 via MQTT ou HTTP para consultar o estado mais recente do dispositivo conhecido 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 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 do 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

AWS IoT Biblioteca de empregos

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

AWS IoT Jobs é um serviço que notifica um ou mais dispositivos conectados sobre 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 AWS IoT Jobs usam o MQTT, um protocolo leve de publicação e assinatura. Essa biblioteca fornece uma API para compor e reconhecer as cadeias de tópicos do MQTT usadas pelo serviço Jobs. AWS IoT

A biblioteca AWS IoT Jobs é escrita em C e projetada 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 do código dos AWS IoT trabalhos (exemplo gerado com o GCC para ARM Cortex-M)		
Arquivo	Com otimização -O1	Com otimização: Os
jobs.c	1,8 K	1,5 K
Estimativas totais	1,8 K	1,5 K

Transport Layer Security

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Visão geral

A biblioteca Wi-Fi do FreeRTOS abstrai as implementações Wi-Fi específicas para a porta em uma API comum que simplifica o desenvolvimento de aplicações e a portabilidade para todas as placas

qualificadas para o FreeRTOS com recursos Wi-Fi. Usando essa API comum, as aplicações podem se comunicar com a pilha sem fio de nível inferior por meio de uma interface comum.

Dependências e requisitos

A biblioteca de Wi-Fi do FreeRTOS exige o núcleo FreeRTOS+TCP.

Atributos

A biblioteca Wi-Fi inclui os seguintes recursos:

- Support para WEP, WPA e WPA2 autenticação WPA3
- 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 o dispositivo estiver no modo AP, não será possível conectar outro dispositivo ao dispositivo do FreeRTOS 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**
>
> As bibliotecas do FreeRTOS não fornecem provisionamento de Wi-Fi no modo AP. Você deve fornecer a funcionalidade adicional, incluindo recursos de servidor DHCP e HTTP, para obter suporte total do 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

A API Wi-Fi oferece suporte aos tipos WEP, WPA e de WPA3 segurança. WPA2 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 Referência da API.

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 API de interface é compatível com 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 Referência da API.

> **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, é necessário inicializar alguns componentes específicos da placa, além dos componentes do FreeRTOS. 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 Referência da API.

Referência da API

Para obter uma referência completa de API, consulte Referência de API do 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 a portabilidade da biblioteca de Wi-Fi, consulte Portabilidade da biblioteca de Wi-Fi no Guia de portabilidade do FreeRTOS.

Demonstrações do FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O FreeRTOS inclui algumas aplicações 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 as aplicações de demonstração, recomendamos que você conclua o tutorial em Conceitos básicos do FreeRTOS. Ele mostra como configurar e executar a demonstração da autenticação mútua do coreMQTT.

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
- Gerenciador de inicialização de demonstração para o Microchip Curiosity MZEF PIC32
- AWS IoT Device Defender demonstração
- AWS IoT Greengrass aplicativo de demonstração de descoberta
- Demonstrações de coreHTTP
- AWS IoT Demo da biblioteca de empregos
- Demonstrações do coreMQTT
- Over-the-air atualiza o aplicativo de demonstração
- Demonstração do cliente Echo de Secure Sockets
- AWS IoT Aplicativo de demonstração Device Shadow

A função `DEMO_RUNNER_RunDemos`, localizada no arquivo *`freertos`*`/demos/demo_runner/iot_demo_runner.c`, inicializa um thread desanexado no qual uma única aplicação de

demonstração é executada. Por padrão, `DEMO_RUNNER_RunDemos` chama e inicia somente a demonstração da autenticação mútua do coreMQTT. 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 uma aplicação 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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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

  Este aplicativo demonstra como usar o APIs middleware FreeRTOS Bluetooth Low Energy 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 o Mobile SDKs usa o Amazon Cognito para autenticação de dispositivos, você não precisa 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 de IDs autenticar seu aplicativo móvel com a 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
   > - Nórdico em RF5284 0-DK

Componentes comuns

As aplicações de demonstração do FreeRTOS têm dois componentes comuns:

- Network Manager
- aplicação de demonstração do SDK móvel de Bluetooth Low Energy

Network Manager

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 estiver Wi-Fi-enabled, o Network Manager mudará para uma conexão Wi-Fi disponível para evitar que você se desconecte da rede.

Para habilitar um tipo de conexão de rede com o Network Manager, adicione o tipo de conexão de rede ao `configENABLED_NETWORKS` parâmetro em `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h` (onde *`vendor`* é o nome do fornecedor e o *`board`* nome da placa que você está usando 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 em Android SDK para dispositivos Bluetooth FreeRTOS abaixo e `amazon-freertos-ble-android-sdk/app` no SDK iOS para dispositivos Bluetooth FreeRTOS em. `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 AWS região e ID da conta.

```
{
    "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 AWS região e ID da conta.

```
{
    "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.

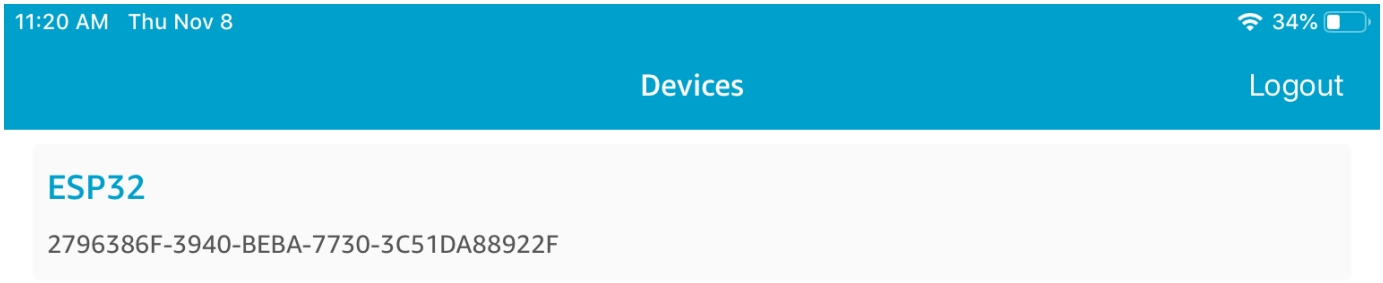
Para descobrir e estabelecer conexões seguras com seu microcontrolador por Bluetooth Low Energy

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:

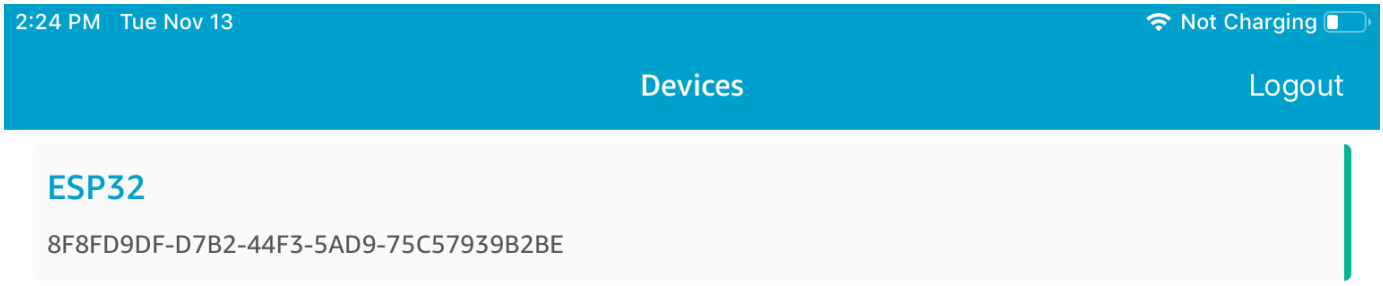
```
$ ./gradlew installDebug
```

4. Confirme se o microcontrolador aparece em Devices (Dispositivos) na aplicação de demonstração do SDK móvel de Bluetooth Low Energy.

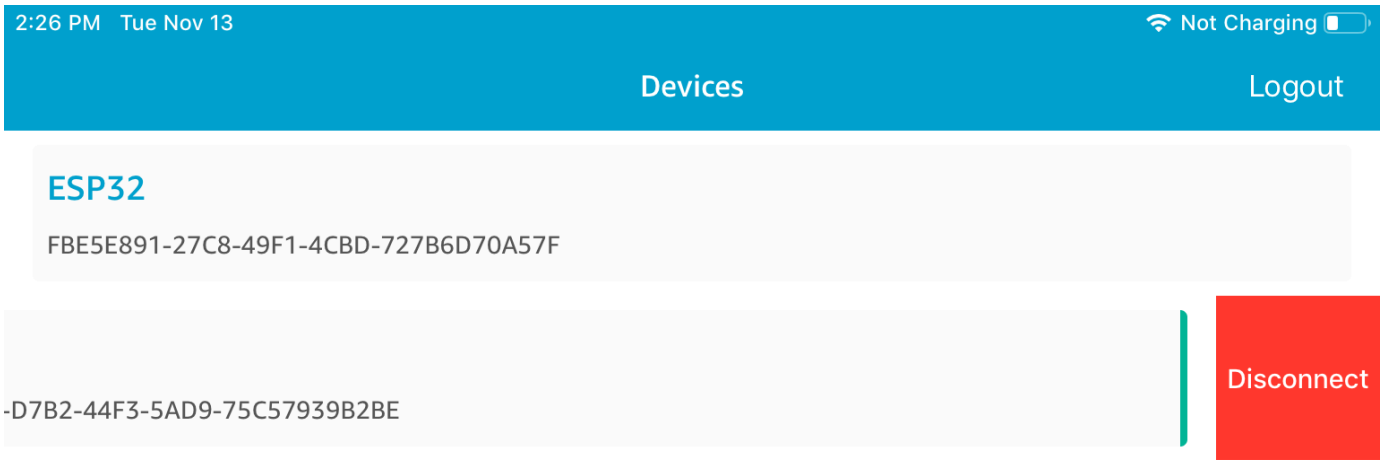


> **Note**
>
> Todos os dispositivos com o FreeRTOS e o serviço de informações de dispositivos (*freertos*`/.../device_information`) que estiverem no intervalo aparecerão na lista.

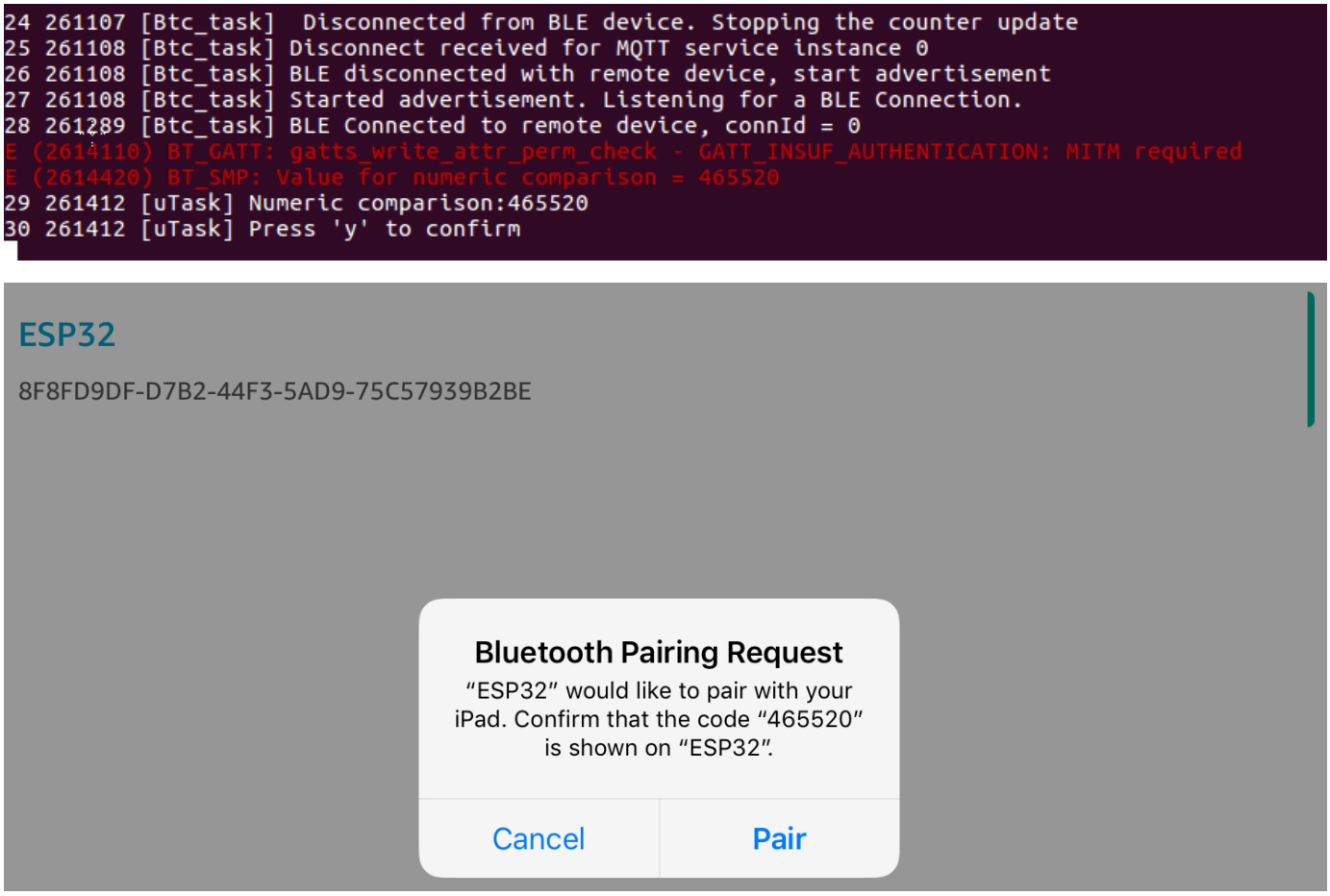
5. Escolha seu microcontrolador na lista de dispositivos. A aplicação estabelece uma conexão com a placa, e uma linha verde é exibida ao lado do dispositivo conectado.



Você pode se desconectar do seu 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 Test (Teste) para abrir o cliente MQTT.
3. Em Tópico de inscrição, insira ***thing-name*/example/topic1** e selecione Inscreva-se no tópico.

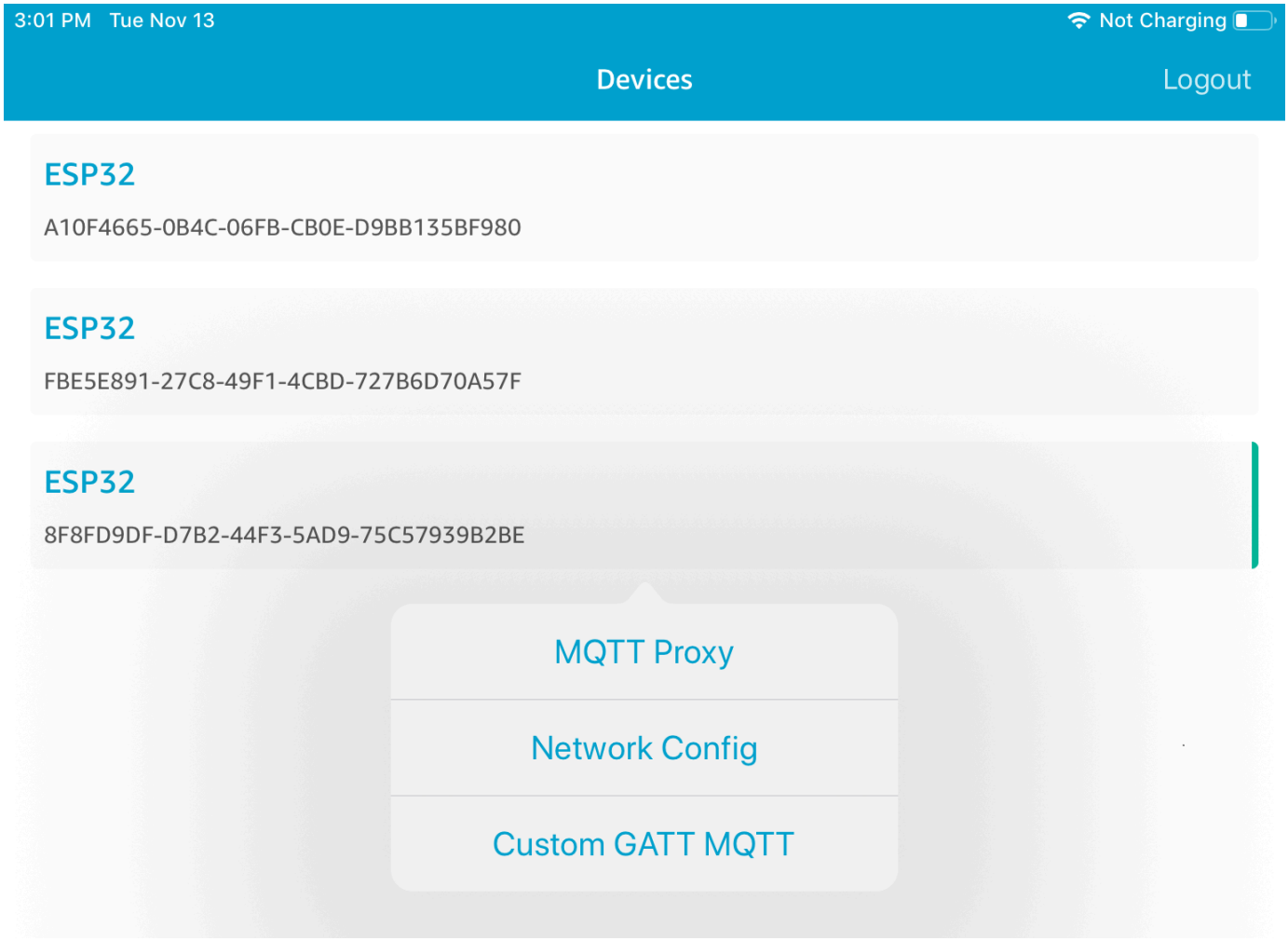
Se você usa Bluetooth Low Energy para emparelhar o microcontrolador com seu dispositivo móvel, as mensagens MQTT são roteadas por meio da aplicação de demonstração do SDK móvel de Bluetooth Low Energy em seu dispositivo móvel.

Como habilitar a demonstração por Bluetooth Low Energy

- Abra `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` e defina `CONFIG_MQTT_BLE_TRANSPORT_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 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 ESP32 Espressif - C. DevKit

Para habilitar a demonstração

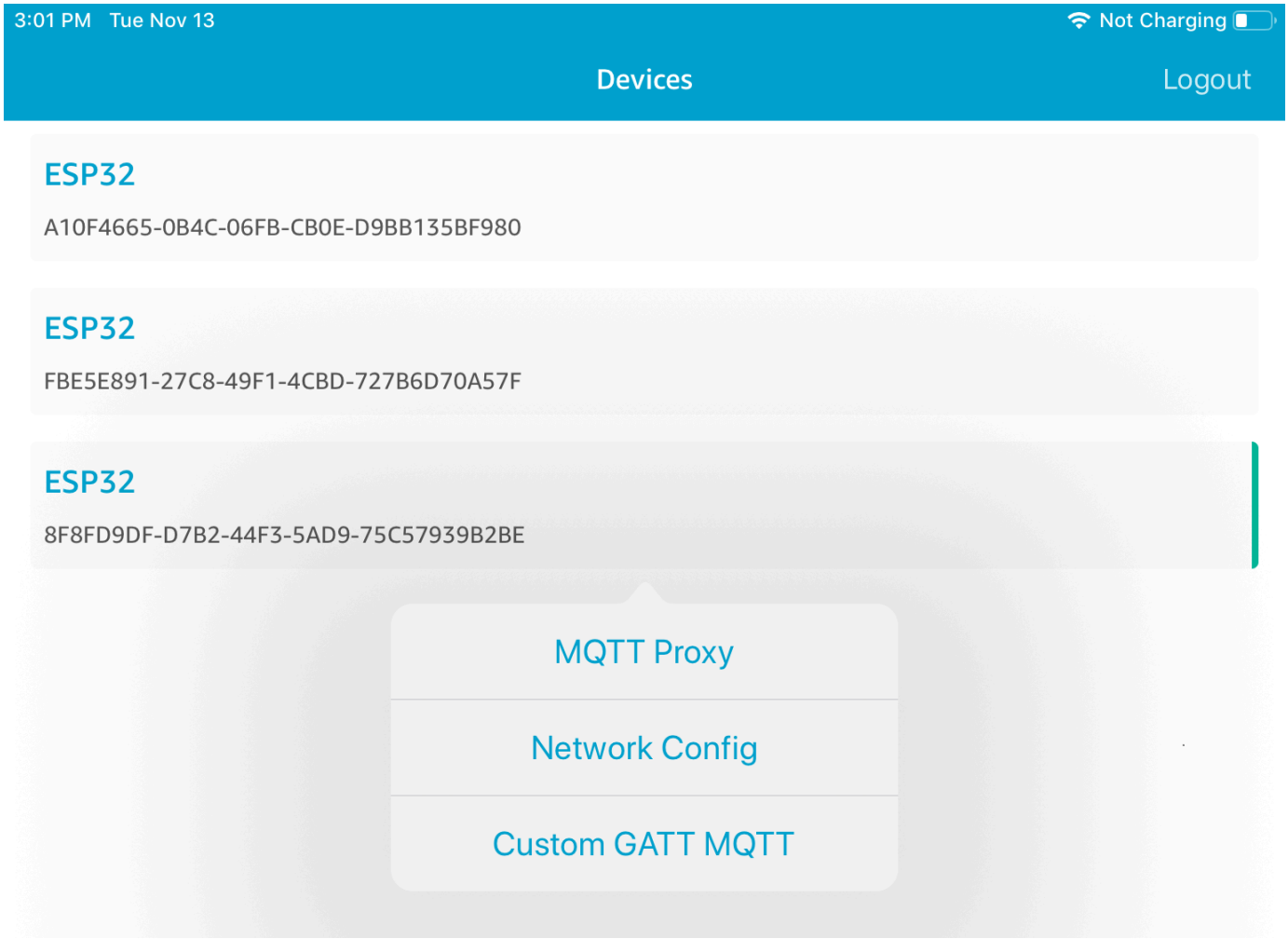
1. Habilite o serviço de provisionamento de Wi-Fi. Abra `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` e `#define IOT_BLE_ENABLE_WIFI_PROVISIONING` defina como 1 (onde *vendor* é o nome do fornecedor e o nome do quadro que você *board* está usando para executar as demonstrações).

   > **Note**
   >
   > O serviço de provisionamento de Wi-Fi é desabilitado por padrão.

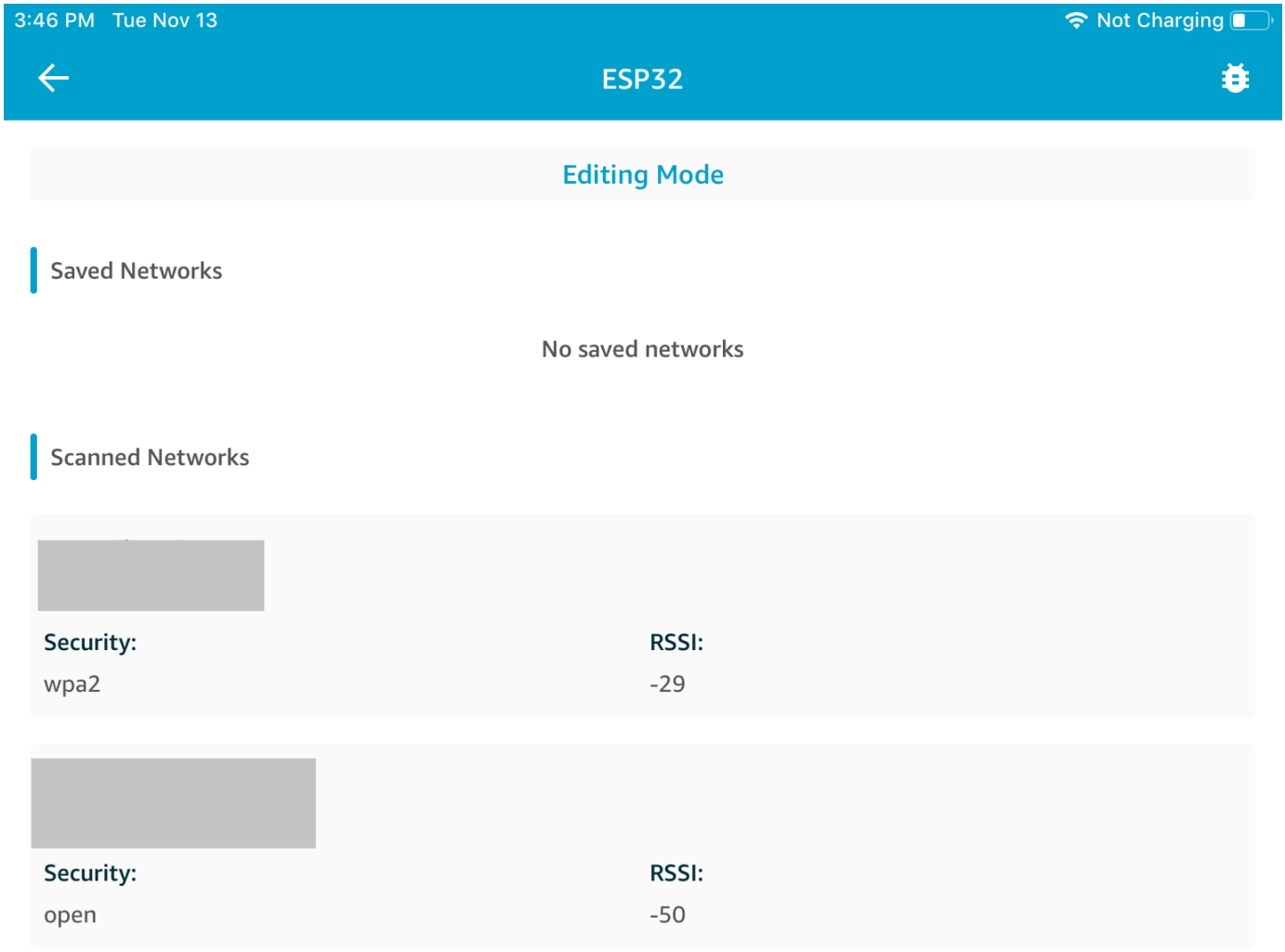
2. Configure o Network Manager para ativar tanto Bluetooth Low Energy quanto Wi-Fi.

Para executar a demonstração

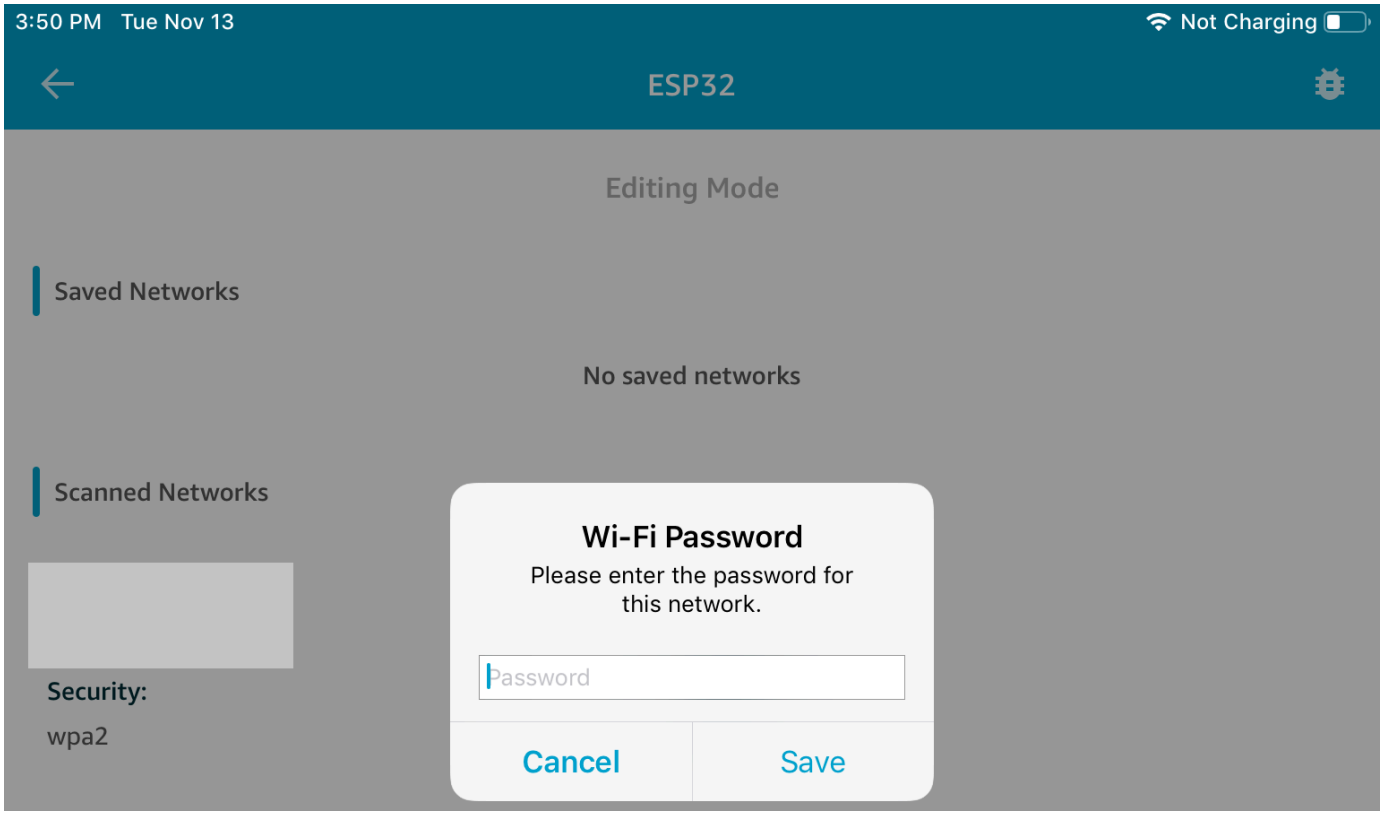
1. Compile e execute o projeto de demonstração no seu microcontrolador.
2. Verifique se você pareou o microcontrolador e o dispositivo móvel usando o aplicação de demonstração do SDK móvel de Bluetooth Low Energy do FreeRTOS.
3. Na lista Devices (Dispositivos) na aplicação móvel de demonstração, escolha seu microcontrolador e, depois, escolha Network Config (Configuração de rede) para abrir as configurações de rede.



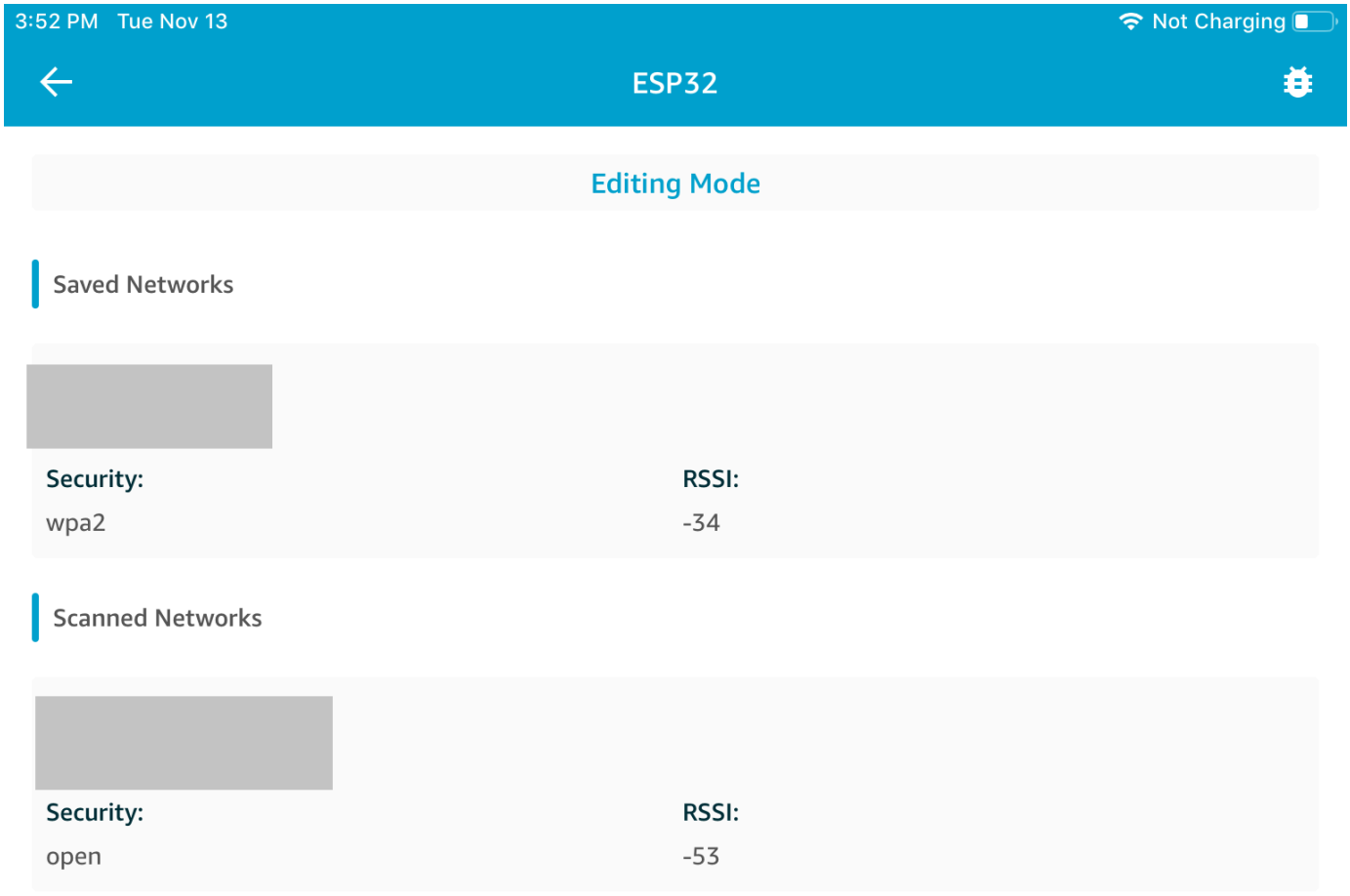
4. Depois que você escolher Network Config (Configuração de rede) para sua placa, o microcontrolador enviará uma lista de redes nas proximidades para o dispositivo móvel. As redes Wi-Fi disponíveis aparecem em uma lista em Scanned Networks (Redes verificadas).



Na lista Scanned Networks (Redes verificadas), escolha a rede e digite o SSID e a senha, se necessário.

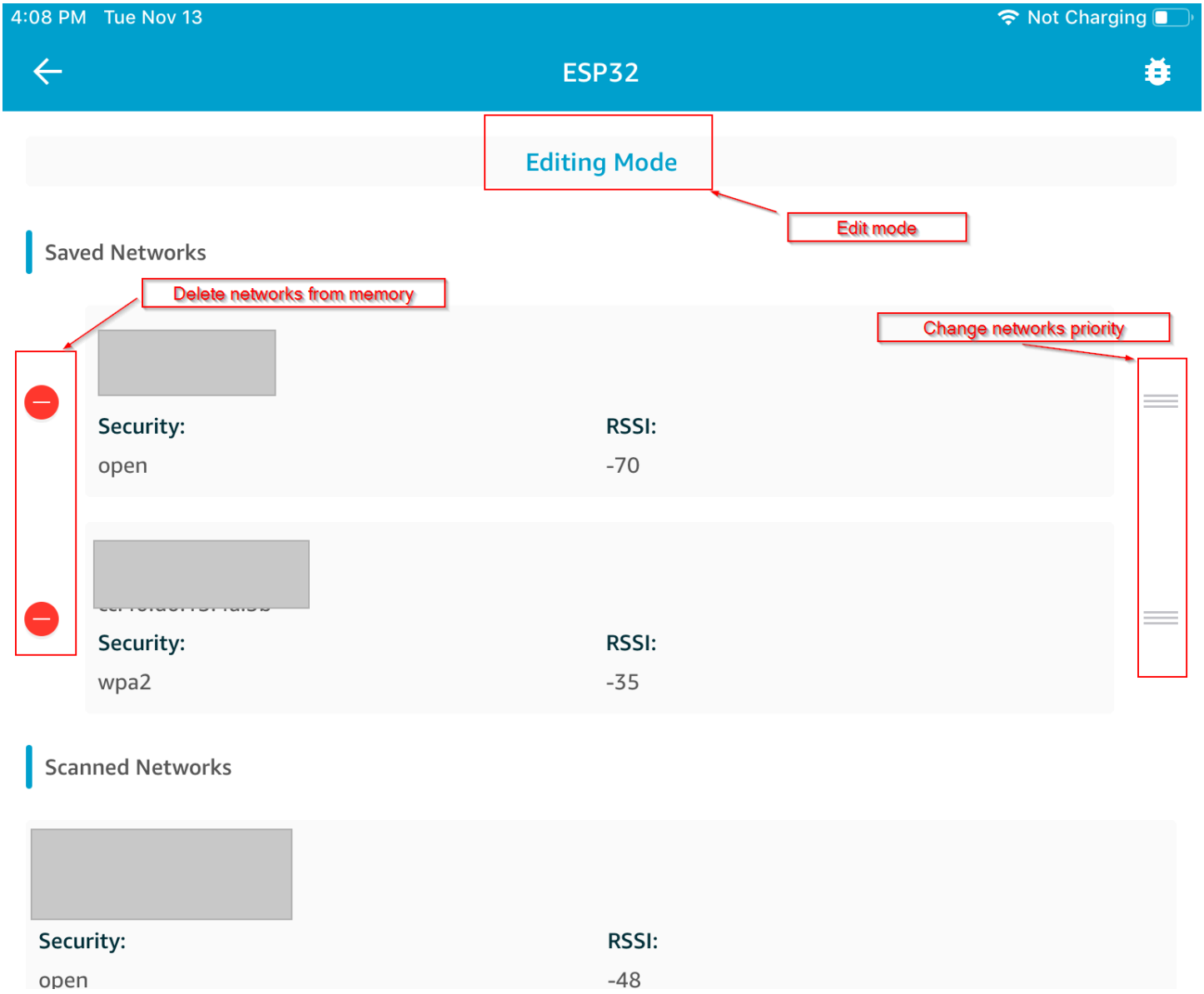


O microcontrolador conecta-se à rede e a salva. A rede é exibida nas Saved Networks (Redes salvas).



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 o Bluetooth Low Energy Mobile SDKs, você pode criar seu próprio cliente GATT para um dispositivo móvel que se conecta ao servidor GATT em seu microcontrolador e é executado em paralelo com o aplicativo móvel de demonstração.

Para habilitar a demonstração

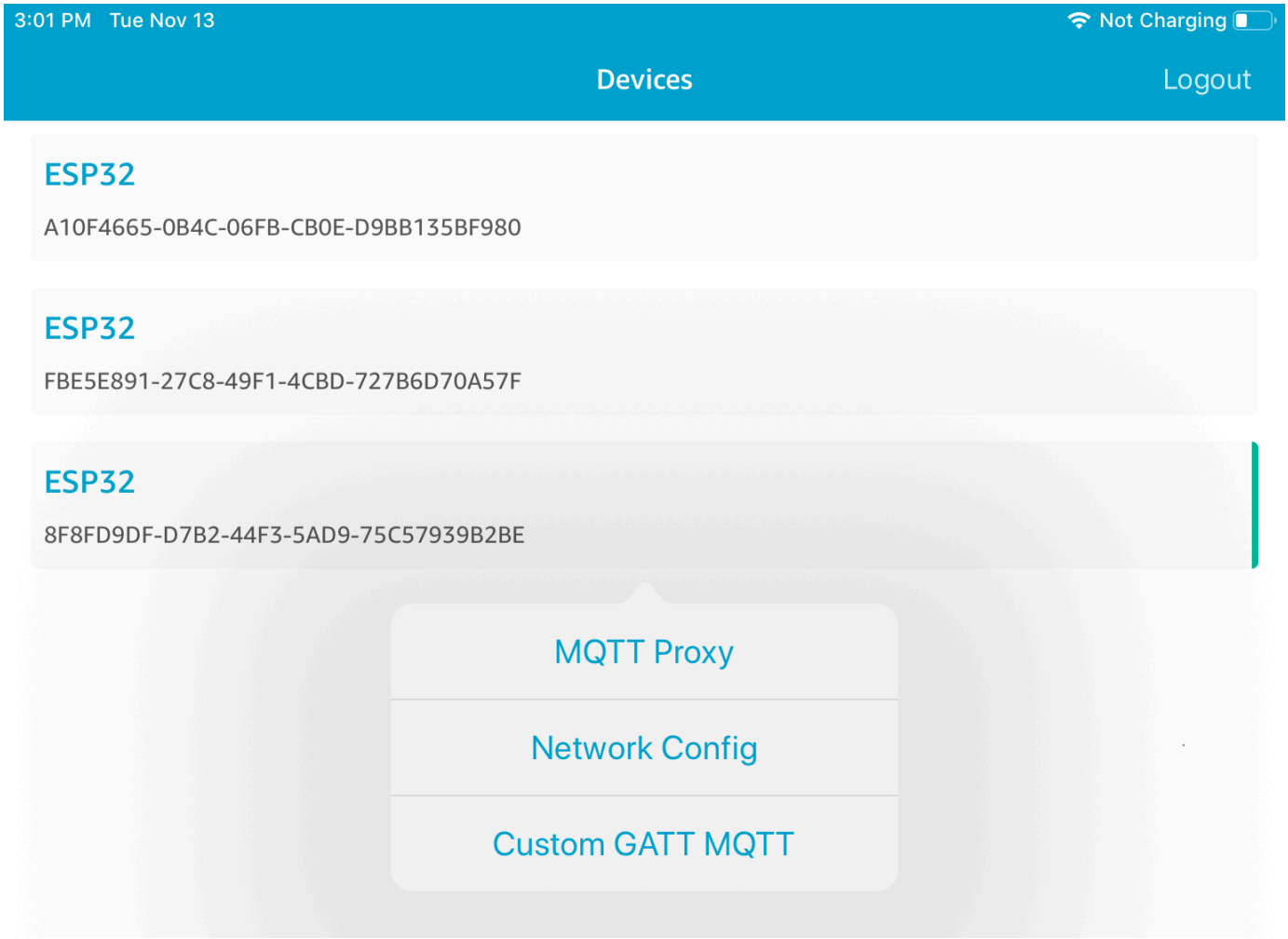
1. Ative a demonstração Bluetooth Low Energy GATT. Em `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` (onde *vendor* é o nome do fornecedor e o nome da placa que você *board* está usando para executar as demonstrações), adicione `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` à lista de declarações definidas.

   > **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_MQTT_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.

Gerenciador de inicialização de demonstração para o Microchip Curiosity MZEF PIC32

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este bootloader de demonstração implementa verificação de versão de firmware, verificação de assinatura criptográfica e testes automáticos de 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.

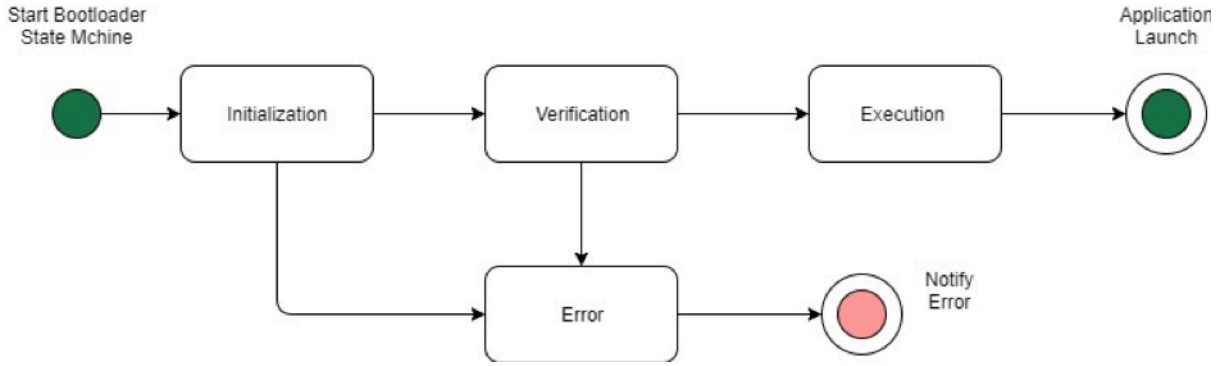
O bootloader é compatível com os seguintes recursos necessários para OTA:

- 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

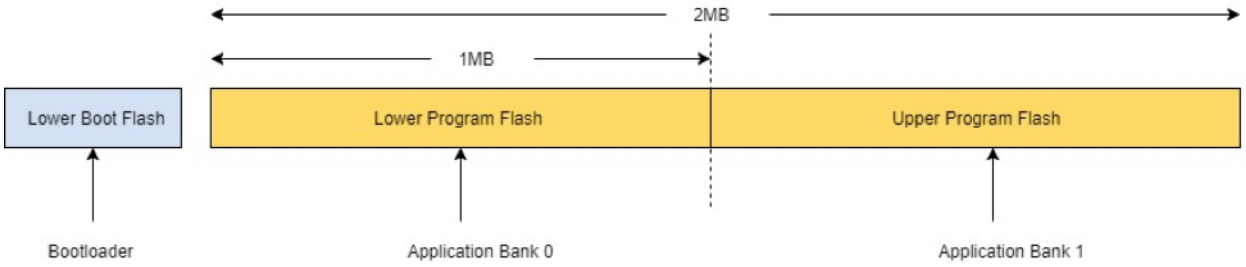
Se a opção de configuração para executar a imagem padrão estiver ativada, o bootloader iniciará a aplicação a partir de um endereço de execução padrão. Essa opção deve ser desativada, exceto durante a depuração.

Estado de erro de bootloader

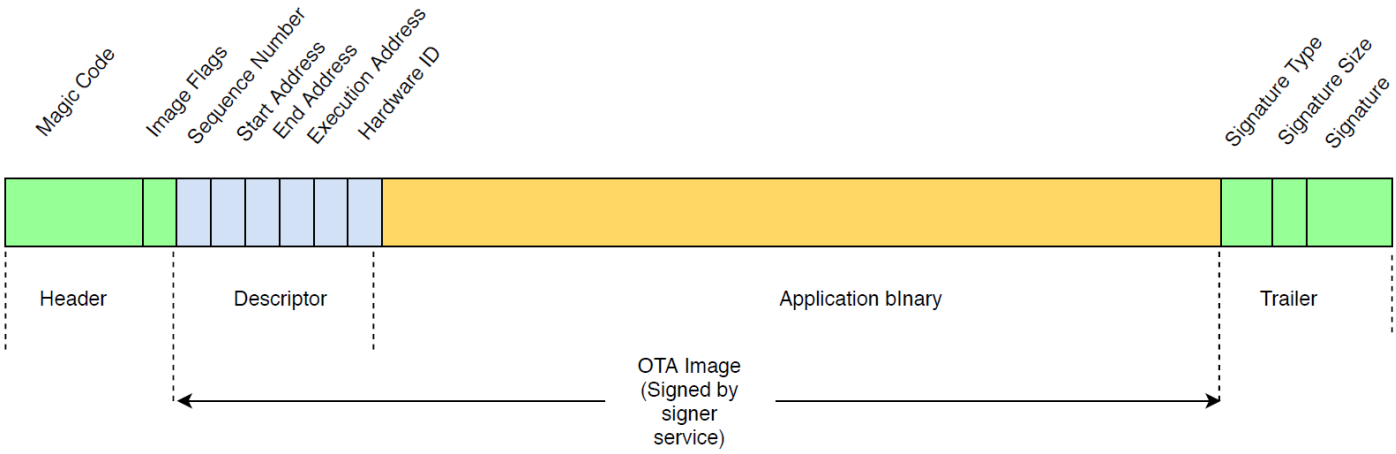
O bootloader está em um estado de erro e nenhuma imagem válida está presente no dispositivo. O bootloader deve notificar o usuário. A implementação padrão envia uma mensagem de log para o console e o LED pisca rapidamente na placa indefinidamente.

Dispositivo flash

A plataforma Microchip Curiosity PIC32 MZEF contém um flash de programa interno 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	Valor	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

Campo do descritor	Tamanho (em 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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

Esta demonstração mostra como usar a biblioteca AWS IoT Device Shadow para se conectar AWS IoT Device Defendera. A demonstração usa a biblioteca CoreMQTT para estabelecer uma conexão MQTT com TLS (autenticação mútua) com o AWS IoT MQTT Broker e a biblioteca CoreJSON para validar e analisar as respostas recebidas do. AWS IoT Device Defender A demonstração mostra como criar um relatório em formato JSON usando métricas coletadas do dispositivo e como enviar o relatório para AWS IoT Device Defender. A demonstração também mostra como registrar uma função de retorno de chamada na biblioteca CoreMQTT para lidar com as respostas AWS IoT Device Defender enviadas para confirmar se um relatório 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 para o AWS IoT Device Defender por meio de uma conexão MQTT segura com o MQTT Broker. AWS IoT

A maneira de coletar métricas depende da pilha TCP/IP em uso. Para as configurações do FreeRTOS+TCP e 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, são fornecidas definições de esboço das funções de coleta de métricas que retornam zeros para todas as métricas. Para enviar métricas reais às placas usando essa implementação de esboço, implemente as funções *freertos*/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c na sua pilha de rede. O arquivo também está disponível no GitHubsite.

Pois ESP32, a configuração padrão do LWIP não usa bloqueio central e, portanto, a demonstração usará métricas fragmentadas. 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.

```
15 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:632] MQTT connection successfully established with broker.


16 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:660] A clean MQTT connection is established. Cleaning up all the
 stored outgoing publishes.


17 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/js
on/accepted to broker.


18 2086 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
19 2086 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.


20 2686 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/js
on/rejected to broker.


21 2746 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
22 2746 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.


23 3346 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:864] the published payload:{"header": {"report_id": 1499,"versio
n": "1.0"},"metrics": {"listening_tcp_ports": {"ports": [{"port": 26023}],"total": 1},"listening_udp_ports": {"ports": [
],"total": 0},"network_stats": {"bytes_in": 0,"bytes_out": 0,"packets_in":
24 3346 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:892] PUBLISH sent for topic $aws/things/MyThingName/defender/met
rics/json to broker with packet ID 3.


25 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
26 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] Ack packet deserialized with result: MQTTSuccess.
27 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] State record updated. New state=MQTTPublishDone.
28 3506 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:484] PUBACK received for packet id 3.


29 3506 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:448] Cleaned up outgoing publish packet with packet id 3.


30 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=145.
31 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSucc
ess.
32 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] State record updated. New state=MQTTPublishDone.
33 3906 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/met
rics/json/accepted to broker.


34 3986 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
35 3986 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.


36 4586 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/met
rics/json/rejected to broker.


37 4646 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
38 4646 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.


39 5246 [iot_thread] [INFO] [MQTT] [core_mqtt.c:2118] Disconnected from the broker.
40 5246 [iot_thread] [INFO ][DEMO][5246] memory_metrics::freertos_heap::before::bytes::2088152
[INFO ][DEMO][5246] memory_metrics::freertos_heap::after::bytes::1986576
41 5246 [iot_thread] [INFO ][DEMO][5246] memory_metrics::demo_task_stack::before::bytes::1908
[INFO ][DEMO][5246] memory_metrics::demo_task_stack::after::bytes::1908
42 6246 [iot_thread] [INFO ][DEMO][6246] Demo completed successfully.
[INFO ][INIT][6248] SDK cleanup done.
43 6248 [iot_thread] [INFO ][DEMO][6248] -------DEMO FINISHED-------
```

Se sua placa não estiver usando FreeRTOS+TCP ou uma configuração lwIP compatível, a saída será semelhante à seguinte.

```
15 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:632] MQTT connection successfully established with broker.


16 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:660] A clean MQTT connection is established. Cleaning up all the
 stored outgoing publishes.


17 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/js
on/accepted to broker.


18 1568 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
19 1568 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.


20 2168 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/js
on/rejected to broker.


21 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
22 2248 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.


23 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:82] Using stub definition of GetNetworkStats! P
lease implement for your network stack to get correct metrics.
24 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:110] Using stub definition of GetOpenTcpPorts!
Please implement for your network stack to get correct metrics.
25 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:146] Using stub definition of GetOpenUdpPorts!
Please implement for your network stack to get correct metrics.
26 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:183] Using stub definition of GetEstablishedCon
nections! Please implement for your network stack to get correct metrics.
27 2848 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:864] the published payload:{"header": {"report_id": 1093,"versio
n": "1.0"},"metrics": {"listening_tcp_ports": {"ports": [],"total": 0},"listening_udp_ports": {"ports": [],"total": 0},"
network_stats": {"bytes_in": 0,"bytes_out": 0,"packets_in": 0,"packets_out"
28 2848 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:892] PUBLISH sent for topic $aws/things/MyThingName/defender/met
rics/json to broker with packet ID 3.


29 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
30 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] Ack packet deserialized with result: MQTTSuccess.
31 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] State record updated. New state=MQTTPublishDone.
32 2968 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:484] PUBACK received for packet id 3.


33 2968 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:448] Cleaned up outgoing publish packet with packet id 3.


34 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=145.
35 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSucc
ess.
36 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] State record updated. New state=MQTTPublishDone.
37 3368 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/met
rics/json/accepted to broker.


38 3408 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
39 3408 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.


40 4008 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/met
rics/json/rejected to broker.


41 4088 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
42 4088 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.


43 4688 [iot_thread] [INFO] [MQTT] [core_mqtt.c:2118] Disconnected from the broker.
44 4688 [iot_thread] [INFO ][DEMO][4688] memory_metrics::freertos_heap::before::bytes::2088152
[INFO ][DEMO][4688] memory_metrics::freertos_heap::after::bytes::1986760
45 4688 [iot_thread] [INFO ][DEMO][4688] memory_metrics::demo_task_stack::before::bytes::1908
[INFO ][DEMO][4688] memory_metrics::demo_task_stack::after::bytes::1908
46 5688 [iot_thread] [INFO ][DEMO][5688] Demo completed successfully.
[INFO ][INIT][5690] SDK cleanup done.
47 5690 [iot_thread] [INFO ][DEMO][5690] -------DEMO FINISHED-------
```

O código-fonte da demonstração está em seu download no *freertos*/demos/device_defender_for_aws/ diretório ou no GitHubsite.

Inscrição em tópicos do 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.

Geração de relatório do AWS IoT Device Defender

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 de AWS IoT Device Defender, 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.

AWS IoT Greengrass aplicativo de demonstração de descoberta

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Antes de executar a demonstração do AWS IoT Greengrass Discovery para FreeRTOS, você precisa AWS configurar AWS IoT Greengrass, e. AWS IoT Para configurar AWS, siga as instruções emConfigurando sua AWS conta e permissões. Para configurar AWS IoT Greengrass, você precisa criar um grupo do Greengrass e depois adicionar um núcleo do Greengrass. Para obter mais informações sobre a configuração AWS IoT Greengrass, consulte Introdução ao AWS IoT Greengrass.

Depois de configurar AWS e AWS IoT Greengrass, você precisa configurar algumas permissões adicionais para AWS IoT Greengrass.

Para configurar AWS IoT Greengrass permissões

1. Navegue até o console do IAM.
2. No painel de navegação, escolha Funções e, em seguida, localize e escolha ServiceRoleGreengrass_.
3. Escolha Anexar políticas, selecione AmazonS3 FullAccess e AWSIoTFullAccess e, em seguida, escolha 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. Escolha Greengrass_ eServiceRole, em seguida, escolha Salvar.

Conecte sua placa AWS IoT e configure sua 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 nova AWS IoT Greengrass política

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 AWS IoT Greengrass permissões a todos os recursos.
5. Escolha Criar.

Para anexar a AWS IoT Greengrass política ao certificado do seu 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 baixando o FreeRTOS do console do FreeRTOS, escolha Conectar a - em vez de Conectar a AWS IoT Greengrass -. ***Platform*** AWS IoT***Platform***

3. Configuração das demonstrações do FreeRTOS.

   Abra *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, comente `#define CONFIG_MQTT_DEMO_ENABLED` e defina `CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED`.

Depois de instalar AWS IoT e AWS IoT Greengrass depois de baixar e configurar os FreeRTOS, você pode criar, atualizar e executar a demonstração do Greengrass em 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 para o núcleo do Greengrass e para o cliente MQTT. AWS IoT Para visualizar as mensagens no cliente AWS IoT MQTT, abra o AWS IoT console, escolha Testar e, em seguida, 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
```

Usando uma EC2 instância da Amazon

Se você estiver trabalhando com uma EC2 instância da Amazon

1. Encontre o DNS público (IPv4) associado à sua EC2 instância da Amazon — acesse o EC2 console da Amazon e, no painel de navegação esquerdo, escolha Instâncias. Escolha sua EC2 instância da Amazon e, em seguida, escolha o painel Descrição. Procure a entrada do Public DNS (IPv4) e anote-a.
2. Encontre a entrada para grupos de segurança e escolha o grupo de segurança anexado à sua EC2 instância da Amazon.

3. Escolha a guia Regras de entrada e escolha Editar regras de entrada e adicione as seguintes regras.

   Regras de entrada

Tipo	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	-
Todo o ICMP - IPv4	ICMP	Todos	0.0.0.0/0	-
Todo o ICMP - IPv4	ICMP	Todos	::0/0	-

4. No AWS IoT console, escolha Greengrass, depois Grupos e escolha o grupo Greengrass que você criou anteriormente. Escolha 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) que você anotou na primeira etapa.
7. Adicione a coisa da IoT do FreeRTOS criada para o grupo do GG.

a. Escolha a seta para voltar para retornar à página do AWS IoT Greengrass grupo. 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	Alvo	Tópico
TIGG1	IoT Cloud	freertos/demos/ggd

   Onde “Fonte” é o nome dado à AWS IoT coisa criada no AWS IoT console quando você registrou sua placa - TIGG1 "" no exemplo dado aqui.

9. Inicie uma implantação do seu AWS IoT Greengrass grupo e certifique-se de que a implantação seja bem-sucedida. Agora você deve conseguir executar com êxito a demonstração da AWS IoT Greengrass descoberta.

Demonstrações de coreHTTP

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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:

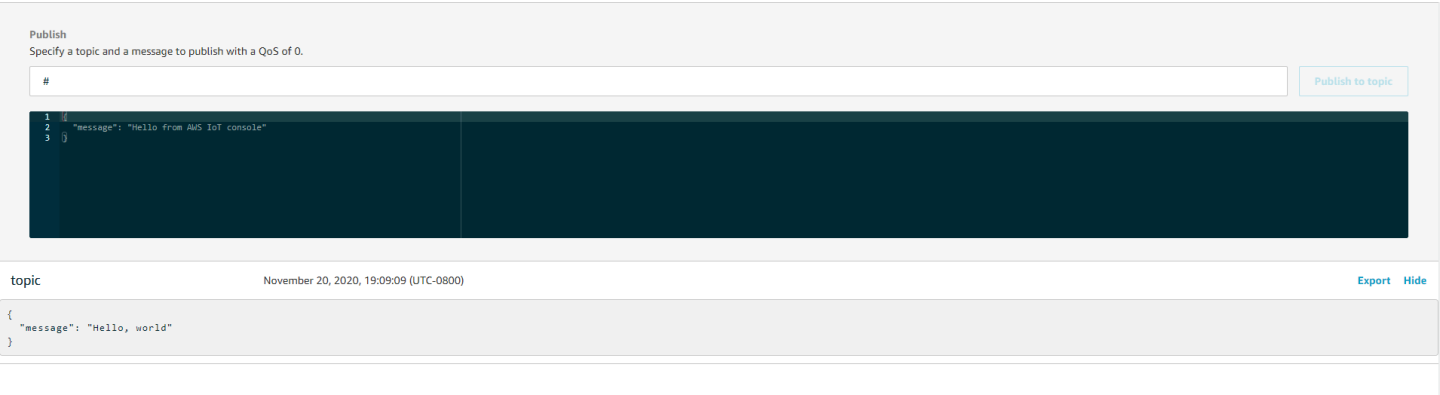
- Conecte-se ao servidor HTTP no AWS IoT endpoint.
- Enviar uma solicitação POST.
- Receber a resposta.
- Desconectar do servidor.

Depois de concluir essas etapas, a demonstração gera uma saída semelhante à da captura de tela a seguir.

```
9 1565 [iot_thread] [INFO ][DEMO][1565] ---------STARTING DEMO---------

10 1566 [iot_thread] [INFO ][INIT][1566] SDK successfully initialized.
11 1566 [iot_thread] [INFO ][DEMO][1566] Successfully initialized the demo. Network type for the demo: 4
12 1566 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:319] 13 1566 [iot_thread] Establishing a TLS session to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com:8443.14 1566 [iot_thread]
15 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (35.166.102.88) will be stored
16 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (35.166.2.12) will be stored
17 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.239.154.164) will be stored
18 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.239.97.33) will be stored
19 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.238.43.70) will be stored
20 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (52.32.144.134) will be stored
21 2042 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:393] 22 2042 [iot_thread] Sending HTTP POST request to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...23 2042 [iot_thread]
24 2082 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:418] 25 2082 [iot_thread] Received HTTP response from a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...
26 2082 [iot_thread]
27 2082 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:283] 28 2082 [iot_thread] Demo completed successfully.29 2082 [iot_thread]
30 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::freertos_heap::before::bytes::2088152
31 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::freertos_heap::after::bytes::1990104
32 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::demo_task_stack::before::bytes::1908
33 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::demo_task_stack::after::bytes::1908
34 3082 [iot_thread] [INFO ][DEMO][3082] Demo completed successfully.
35 3084 [iot_thread] [INFO ][INIT][3084] SDK cleanup done.
36 3084 [iot_thread] [INFO ][DEMO][3084] -------DEMO FINISHED------
```

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 connectToServerWithBackoffRetriesfunção tenta fazer uma conexão TLS autenticada mutuamente com o AWS IoT servidor HTTP. 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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 você 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 do IAM 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_generato/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 connectToServerWithBackoffRetriesfunção 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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 do IAM 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_generato/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. Ele usa a interface de transporte baseada em mbedTLS que é implementada no arquivo FreeRTOS- _mbedtls.c. Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using

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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 de 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 -> RTOSDemos Propriedades -> Conjunto de ferramentas de 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.

AWS IoT Demo da biblioteca de empregos

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

A demonstração da biblioteca AWS IoT Jobs mostra como se conectar ao serviço AWS IoT Jobs por meio de uma conexão MQTT, recuperar um trabalho e processá-lo em um dispositivo. AWS IoT O projeto de demonstração de AWS IoT Jobs usa a porta FreeRTOS para Windows, para que possa ser criado 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 corretor AWS IoT MQTT usando TLS da mesma forma que a demonstração de autenticação mútua do MQTT.

> **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 `jobs_demo.c` arquivo e pode ser encontrado no GitHubsite ou no *`freertos`*`/demos/jobs_for_aws/` diretório.

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 na demonstração de autenticação mútua do MQTT.

Funcionalidade

A demonstração mostra o fluxo de trabalho usado para receber trabalhos AWS IoT e processá-los em um dispositivo. A demonstração é interativa e exige que você crie trabalhos usando o AWS IoT console 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 o 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 da Jobs

O código-fonte da função de ponto de entrada de demonstração do Jobs pode ser encontrado em GitHub. Essa função executa as seguintes operações:

1. Estabeleça uma conexão MQTT usando as funções auxiliares em `mqtt_demo_helpers.c`.
2. Inscreva-se no tópico MQTT para a API `NextJobExecutionChanged`, usando funções auxiliares em `mqtt_demo_helpers.c`. A string do tópico é montada anteriormente, usando macros definidas pela biblioteca AWS IoT Jobs.
3. Publique no tópico MQTT para a API `StartNextPendingJobExecution`, usando funções auxiliares em `mqtt_demo_helpers.c`. A string do tópico é montada anteriormente, usando macros definidas pela biblioteca AWS IoT Jobs.
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 prvEventCallbackfunção chama a `Jobs_MatchTopic` partir da biblioteca AWS IoT Jobs para classificar a mensagem MQTT recebida. 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**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Essas demonstrações podem ajudar você a aprender como usar a biblioteca coreMQTT.

Tópicos

- Demonstração de autenticação mútua da coreMQTT
- Demonstração de compartilhamento de conexão coreMQTT

Demonstração de autenticação mútua da coreMQTT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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.

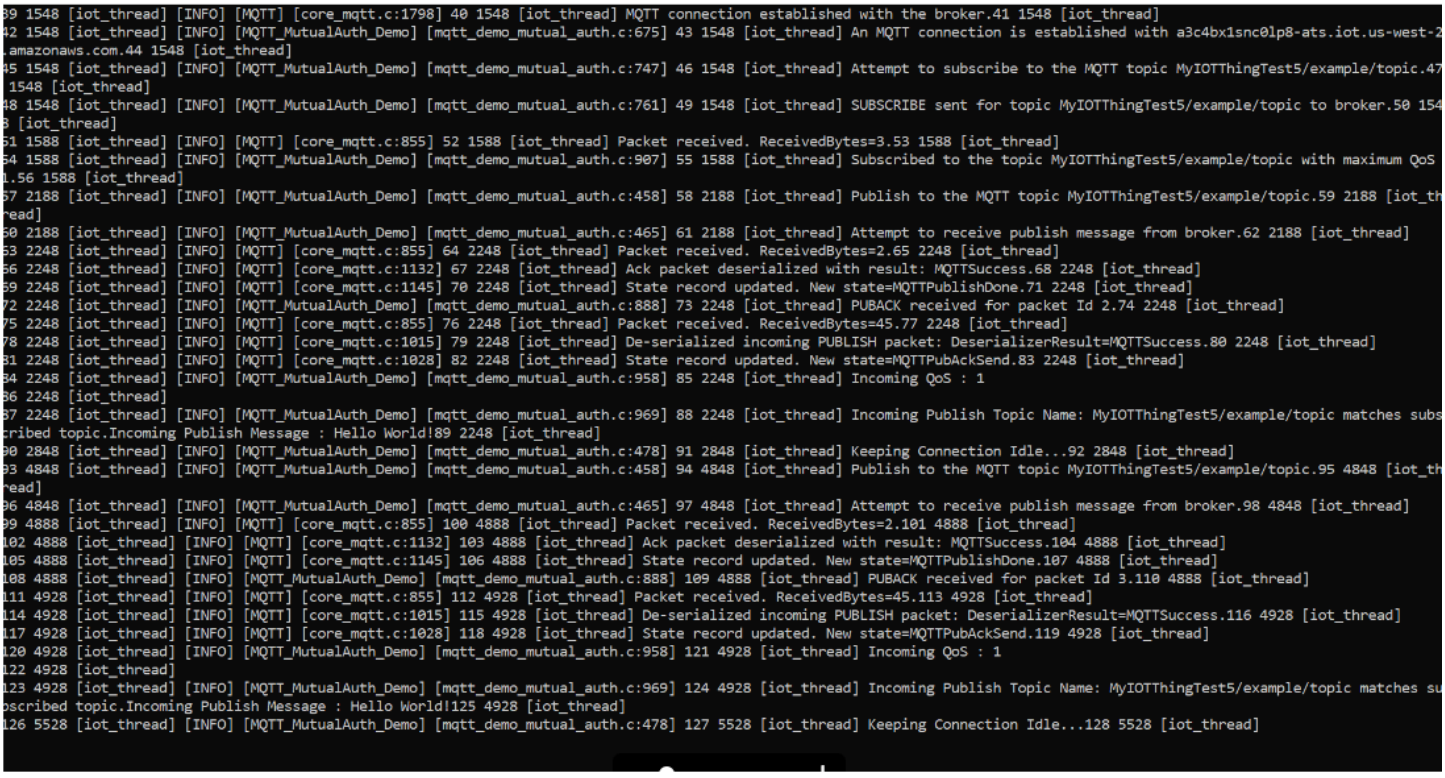
Código-fonte

O arquivo fonte de demonstração tem um nome `mqtt_demo_mutual_auth.c` e pode ser encontrado no *`freertos`*`/demos/coreMQTT/` diretório e no GitHubsite.

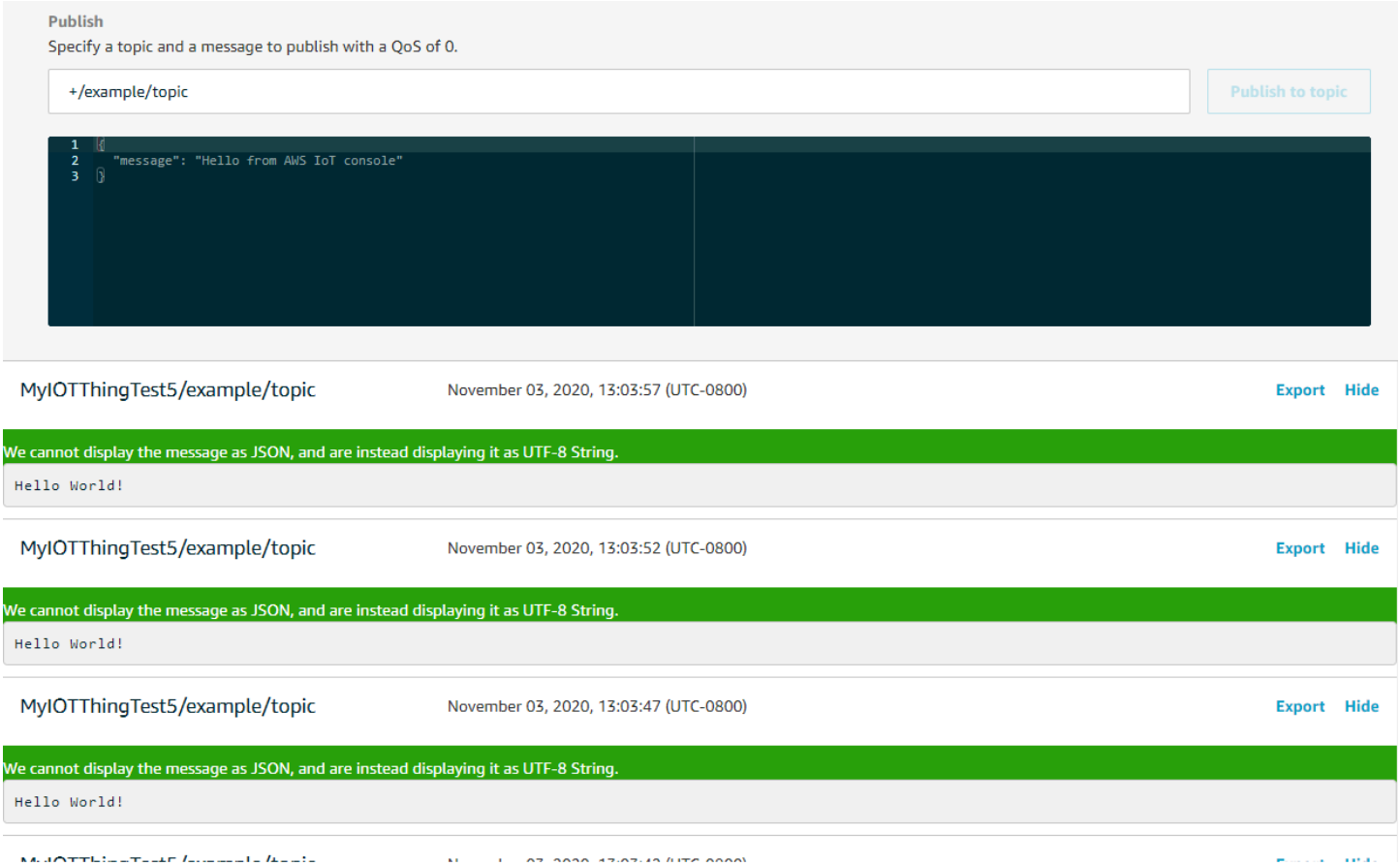
Funcionalidade

A demonstração cria uma única tarefa de aplicação que percorre um conjunto de exemplos demonstrando como se conectar ao agente, assinar um tópico no agente, publicar em um tópico no agente e, enfim, se desconectar do agente. A aplicação 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 a aplicação de demonstração.

A conclusão com êxito da demonstração gerará um resultado semelhante ao da imagem a seguir.



O AWS IoT console gerará uma saída semelhante à imagem a seguir.



Lógica de repetição com recuo exponencial e jitter

A função prvBackoffForRetry mostra como operações de rede com falha com o servidor, por exemplo, conexões TLS ou solicitações de assinatura MQTT, podem ser repetidas com recuo e instabilidade exponenciais. 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 requer a geração de um número aleatório, a função usa o PKCS11 módulo para gerar o número aleatório. O uso do PKCS11 módulo permite o acesso a um Gerador de Números Aleatórios Verdadeiros (TRNG) se a plataforma do fornecedor o suportar. 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 prvConnectToServerWithBackoffRetriesfunção 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 MQTTConnection WithBroker função prvCreate 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 MQTTSubscribe WithBackoffRetries função prv demonstra como assinar um filtro de tópicos no broker 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 MQTTPublish ToTopic função prv demonstra como publicar em um tópico no broker MQTT.

Recebimento de mensagens

A aplicação 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 pela aplicação. A prvEventCallbackfunção é um exemplo dessa função de retorno de chamada de evento. `prvEventCallback`examina o tipo de pacote de entrada 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 MQTTProcess IncomingPublish função prv 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`. Aqui está a definição da função prv MQTTUnsubscribe FromTopic.

Alteração da CA raiz usada na demonstração

Por padrão, as demonstrações do FreeRTOS usam o certificado Amazon Root CA 1 (chave RSA de 2048 bits) para se autenticar no servidor. 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 coreMQTT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

O projeto de demonstração do compartilhamento de conexão CoreMQTT mostra como usar um aplicativo multiencadeado para estabelecer uma conexão com o agente AWS 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. A demonstração assina 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 quatro tarefas nesta demonstração.

- Uma tarefa de comando (principal) recebe comandos da fila de comandos e os processa. As outras tarefas colocam comandos nessa fila para serem processados. Essa tarefa entra em um loop, durante o qual processa os comandos. Se um comando de encerramento for recebido, essa tarefa escapará do loop.
- Uma tarefa de publicador síncrona cria uma série de operações de publicação e as envia para a fila de comandos. Em seguida, essas operações são executadas pela tarefa de comando. Essa tarefa usa publicação síncrona, o que significa que ela aguardará a conclusão de cada operação de publicação antes de programar a próxima.
- Uma tarefa de publicador assíncrona cria uma série de operações de publicação e as envia para a fila de comandos. Em seguida, essas operações são executadas pela tarefa de comando. A diferença entre essa tarefa e a anterior é que ela não espera a conclusão de uma operação de publicação antes de programar a próxima. Ela verifica o status de cada operação de publicação após todas as operações de publicação serem adicionadas à fila. Observe que a diferença entre a publicação síncrona e assíncrona está apenas no comportamento dessas tarefas. Não há diferenças nos comandos de publicação propriamente ditos.
- Uma tarefa de assinante cria uma assinatura MQTT em um filtro de tópicos que corresponde aos tópicos de todas as mensagens publicadas pelas duas tarefas do publicador. Essa tarefa entra em um loop e espera receber as mensagens que foram publicadas pelas outras tarefas.

As tarefas podem ter filas para manter as mensagens recebidas. A tarefa de comando enviará as mensagens recebidas para a fila de qualquer tarefa assinada no tópico de entrada.

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, o com thread única e o com threads múltiplas (multitarefas). Esta demonstração mostra como você pode criar o próprio esquema com threads múltiplas. Há também outro exemplo com threads múltiplas que executa o protocolo MQTT em segundo plano em uma tarefa de agente (ou daemon). Para obter mais informações, consulte Agente e demonstrações MWTT usando a coreMQTT. Se você executar o protocolo MQTT em uma tarefa do agente, não será necessário gerenciar explicitamente nenhum estado do MQTT ou chamar a função `MQTT_ProcessLoop`. Além disso, se você usar uma tarefa de agente, várias tarefas de

aplicação poderão compartilhar uma única conexão MQTT sem a necessidade de primitivas de sincronização, como mutexes.

Código-fonte

O arquivo fonte de demonstração tem um nome `mqtt_demo_connection_sharing.c` e pode ser encontrado no *`freertos`*`/demos/coreMQTT/` diretório e no GitHubsite.

Funcionalidade

Essa demonstração cria quatro tarefas no total: três que solicitam chamadas à API MQTT e uma que processa essas solicitações e é a tarefa principal. Nesta demonstração, a tarefa principal entra em um loop que cria as três subtarefas, chama o loop de processamento e depois faz a limpeza. A tarefa principal cria uma conexão MQTT única com o agente que é compartilhada entre as subtarefas. Duas das subtarefas publicam mensagens para o agente e a terceira recebe as mensagens de volta, usando uma assinatura MQTT para um filtro de tópicos que corresponde a todos os tópicos das mensagens publicadas.

Typedefs

A demonstração define as seguintes estruturas, enumerações e ponteiros de função.

Comandos

Em vez de fazer chamadas diretamente à API MQTT, as tarefas usam estruturas `Command_t` para criar comandos que instruem a tarefa principal a chamar a operação de API apropriada para elas. Os comandos têm os tipos a seguir:

- `PROCESSLOOP`
- `PUBLISH`
- `SUBSCRIBE`
- `UNSUBSCRIBE`
- `PING`
- `DISCONNECT`
- `RECONNECT`
- `TERMINATE`

O comando `TERMINATE` não tem uma operação de API MQTT correspondente. Ele é usado na demonstração para instruir a tarefa principal a interromper o processamento de

comandos e iniciar as operações de limpeza. Como algumas informações adicionais, por exemplo, informações de publicação ou assinatura, são necessárias para alguns comandos MQTT como `MQTT_Publish`, `MQTT_Subscribe` e `MQTT_Unsubscribe`, usamos o campo `CommandContext_t`. Esse campo é obrigatório para esses três comandos e é opcional para outros.

Como esse contexto é obrigatório para esses comandos, não altere esse valor depois que o comando for colocado na fila, até que o comando seja concluído. Quando um comando é concluído, um retorno de chamada opcional pode ser invocado. Nesta demonstração, usamos um retorno de chamada que cria uma notificação de tarefa para informar à tarefa de chamada que o comando foi concluído. Para operações MQTT que exigem confirmações (assinaturas, cancelamentos de assinaturas e publicações com QoS maior que 0), o comando será considerado concluído quando a confirmação for recebida. Caso contrário, o comando será concluído quando a chamada de API MQTT correspondente for retornada.

As definições a seguir podem ser encontradas no arquivo `mqtt_demo_connection_sharing.c`:

- Estrutura _Command_t
- CommandContextestrutura _t
- CommandType_t enumeração
- CommandCallbackponteiro de função _t

Confirmações

Como algumas operações do MQTT exigem uma confirmação, elas usam uma matriz de AckInfo_t que contém o identificador do pacote da confirmação esperada e o comando original que a espera, para que seu retorno de chamada de conclusão possa ser invocado.

Assinaturas

Essa demonstração pode rastrear assinaturas para cada tarefa. Para fazer isso, cada tarefa que solicita uma assinatura deve fornecer uma fila de mensagens na qual receberá de volta as mensagens publicadas (SubscriptionElement_t). Várias tarefas podem assinar o mesmo filtro de tópicos, pois espera-se que usem filas de resposta separadas.

Mensagens publicadas recebidas

Como as tarefas são executadas paralelamente à tarefa principal, seria difícil e demorado se a tarefa principal tivesse que esperar cada tarefa assinada ler uma mensagem publicada recebida. Portanto, cada mensagem recebida é copiada para a fila de respostas de qualquer tarefa que

esteja inscrita no tópico da mensagem publicada (PublishElement_t). Como os pacotes de publicação recebidos do cliente MQTT contêm ponteiros para o buffer de rede do cliente, a carga e o nome do tópico da mensagem recebida são copiados em buffers separados antes de serem inseridos em uma fila de resposta. Dessa forma, a tarefa assinada ainda poderá ler as informações recebidas após o cliente MQTT limpar o buffer de rede.

Tarefa principal

A tarefa principal do aplicativo, RunCoreMQTTConnectionSharingDemo, estabelece uma sessão MQTT persistente, cria três subtarefas e executa o loop de processamento até que um comando de encerramento seja recebido. Uma sessão persistente será usada, pois se a rede for desconectada inesperadamente, a demonstração se reconectará ao agente em segundo plano, sem perder assinaturas ou mensagens publicadas recebidas do agente. Para criar uma nova sessão persistente para cada execução, a demonstração se conecta ao agente com o sinalizador `clean session` definido, depois se desconecta e se reconecta com o sinalizador não definido. Após o loop de processamento encerrar, ele se desconecta do agente e faz loop novamente a partir do ponto em que fez a reconexão da rede.

A conclusão da demonstração com êxito gerará um resultado semelhante ao da imagem a seguir.

```
128 11020 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 129 11020 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
130 11020 [AsyncPublisher]
131 11036 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 132 11036 [MQTTDemo] Processed 2 operations.133 11036 [MQTTDemo]
134 11036 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 135 11036 [MQTTDemo] Publishing message to filter/sync/1.136 11036 [MQTTDemo]
137 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 138 11056 [MQTTDemo] Processed 3 operations.139 11056 [MQTTDemo]
140 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 141 11056 [MQTTDemo] Publishing message to filter/async/1.142 11056 [MQTTDemo]
143 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvHandleIncomingPublish:1571] 144 11056 [MQTTDemo] Adding publish to response queue for filter/+/+
145 11056 [MQTTDemo]
146 11056 [Subscriber] [INFO] [MQTTDemo] [prvSubscribeTask:2059] 147 11056 [Subscriber] Received publish on topic filter/sync/1
Message payload: Hello World! Sync: 1
148 11056 [Subscriber]
149 11056 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1888] 150 11057 [SyncPublisher] Publish operation complete. Sleeping for 100 ms.
151 11057 [SyncPublisher]
152 11122 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1954] 153 11122 [AsyncPublisher] Adding publish operation for message Hello World! Async: 4
on topic filter/async/4154 11122 [AsyncPublisher]
155 11122 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 156 11122 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
157 11122 [AsyncPublisher]
158 11157 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1873] 159 11157 [SyncPublisher] Adding publish operation for message Hello World! Sync: 2
on topic filter/sync/2160 11157 [SyncPublisher]
161 11157 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1878] 162 11157 [SyncPublisher] Waiting for publish 2 to complete.163 11157 [SyncPublisher]
164 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 165 11196 [MQTTDemo] Processed 4 operations.166 11196 [MQTTDemo]
167 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 168 11196 [MQTTDemo] Publishing message to filter/async/2.169 11196 [MQTTDemo]
170 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvHandleIncomingPublish:1571] 171 11196 [MQTTDemo] Adding publish to response queue for filter/+/+
172 11196 [MQTTDemo]
173 11196 [Subscriber] [INFO] [MQTTDemo] [prvSubscribeTask:2059] 174 11196 [Subscriber] Received publish on topic filter/async/1
Message payload: Hello World! Async: 1
175 11196 [Subscriber]
176 11222 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1954] 177 11222 [AsyncPublisher] Adding publish operation for message Hello World! Async: 5
on topic filter/async/5178 11222 [AsyncPublisher]
179 11222 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 180 11222 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
```

Loop de comando

O loop de comando, prvCommandLoop, espera que os comandos sejam colocados na fila de comandos e, em seguida, chama a API MQTT apropriada. Todos os comandos, exceto `DISCONNECT` e `TERMINATE`, resultam na chamada de `MQTT_ProcessLoop`. Esta demonstração

define um retorno de chamada de ativação do soquete para adicionar um comando `PROCESSLOOP` à fila quando os dados estão disponíveis no soquete. No entanto, podem existir muitos comandos à frente na fila nesse momento. Para garantir que não negligenciemos os dados recebidos enquanto outros comandos são processados, `MQTT_ProcessLoop` é chamado para uma iteração única após cada comando.

Comandos de processamento

Veja a `prvProcessCommand`função.

Tarefa síncrona do publicador

A tarefa síncrona do editor, `prvSyncPublishTarefa`, cria `PUBLISH` operações de forma síncrona e espera que cada operação seja concluída antes de programar a próxima. Esta demonstração usa QoS 1 para publicar mensagens, isso significa que essas operações não são consideradas concluídas até que o pacote de confirmação de publicação seja recebido.

Tarefa de publicador assíncrona

A tarefa assíncrona do editor, `prvAsyncPublishTarefa`, não espera a conclusão de uma publicação antes de colocar a próxima na fila. Isso demonstra que nem sempre é necessário que uma tarefa aguarde a conclusão de uma operação MQTT para poder ser retomada. Como cada comando de publicação exige a própria estrutura de contexto, essa tarefa não pode reutilizar uma estrutura de contexto única como faz a tarefa síncrona do publicador, porque um comando anterior ainda pode precisar dela. Portanto, ela aloca memória para cada estrutura de contexto e, em seguida, espera para liberar toda a memória alocada após todas as mensagens que serão publicadas serem colocadas na fila.

Tarefa do assinante

A tarefa do assinante, `prvSubscribeTask`, assina um filtro de tópicos que corresponde a todos os tópicos das mensagens publicadas nas tarefas síncronas e assíncronas. Em seguida, ela espera receber todas as mensagens publicadas antes de cancelar a assinatura. Essa tarefa também é responsável por criar a operação `TERMINATE` que sinaliza a tarefa principal para encerrar o loop de comando.

Over-the-air atualiza o aplicativo de demonstração

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O FreeRTOS inclui um aplicativo de demonstração que demonstra a funcionalidade da over-the-air biblioteca (OTA). A aplicação de demonstração OTA está localizada no arquivo *freertos*/demos/ota/aws_iot_ota_update_demo.c.

A aplicação 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. Conecta-se ao corretor AWS IoT MQTT 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, a aplicação suspenderá o agente OTA, tentará se reconectar usando atraso exponencial com jitter e, em seguida, reiniciará o agente OTA.

Antes de usar as atualizações OTA, conclua todos os pré-requisitos no Atualizações do FreeRTOS Over-the-Air

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 CC322 0SF-LAUNCHXL

- Microchip Curiosity PIC32 MZEF
- Expressivo ESP32
- Baixe, crie, atualize e execute a demonstração do FreeRTOS OTA no Renesas N RX65

Depois de criar, atualizar e executar o aplicativo de demonstração OTA em seu dispositivo, você pode usar o AWS IoT console ou o AWS CLI para criar um trabalho de atualização do 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.

```
313 267848 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
314 268733 [OTA Task] [OTA] Set job doc parameter [ jobId:
 fe18c7ec_8c31_4438_b0b9_ad55acd95610 ]
315 268734 [OTA Task] [OTA] Set job doc parameter [ streamname: 327 ]
316 268734 [OTA Task] [OTA] Set job doc parameter [ filepath: /sys/mcuflashimg.bin ]
317 268734 [OTA Task] [OTA] Set job doc parameter [ filesize: 130388 ]
318 268735 [OTA Task] [OTA] Set job doc parameter [ fileid: 126 ]
319 268735 [OTA Task] [OTA] Set job doc parameter [ attr: 0 ]
320 268735 [OTA Task] [OTA] Set job doc parameter [ certfile: tisigner.crt.der ]
321 268737 [OTA Task] [OTA] Set job doc parameter [ sig-sha1-rsa:
 Q56qxHRq3Lxv6KkorvilVs4AyGJbWsJd ]
322 268737 [OTA Task] [OTA] Job was accepted. Attempting to start transfer.
323 268737 [OTA Task] Sending command to MQTT task.
324 268737 [MQTT] Received message 50000 from queue.
325 268848 [OTA] [OTA] Queued: 2   Processed: 1   Dropped: 0
326 269039 [MQTT] MQTT Subscribe was accepted. Subscribed.
327 269039 [MQTT] Notifying task.
328 269040 [OTA Task] Command sent to MQTT task passed.
329 269041 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/streams/327

330 269848 [OTA] [OTA] Queued: 2   Processed: 1   Dropped: 0
... // Output removed for brevity
346 284909 [OTA Task] [OTA] file token: 74594452
.. // Output removed for brevity
363 301327 [OTA Task] [OTA] file ready for access.
364 301327 [OTA Task] [OTA] Returned buffer to MQTT Client.
365 301328 [OTA Task] Sending command to MQTT task.
366 301328 [MQTT] Received message 60000 from queue.
367 301328 [MQTT] Notifying task.
```

```
368 301329 [OTA Task] Command sent to MQTT task passed.
369 301329 [OTA Task] [OTA] Published file request to $aws/bin/things/TI-LaunchPad/
streams/327/get
370 301632 [OTA Task] [OTA] Received file block 0, size 1024
371 301647 [OTA Task] [OTA] Remaining: 127
... // Output removed for brevity
508 304622 [OTA Task] Sending command to MQTT task.
509 304622 [MQTT] Received message 70000 from queue.
510 304622 [MQTT] Notifying task.
511 304623 [OTA Task] Command sent to MQTT task passed.
512 304623 [OTA Task] [OTA] Published file request to $aws/bin/things/TI-LaunchPad/
streams/327/get
513 304860 [OTA] [OTA] Queued: 47   Processed: 47   Dropped: 83
514 304926 [OTA Task] [OTA] Received file block 4, size 1024
515 304941 [OTA Task] [OTA] Remaining: 82
... // Output removed for brevity
797 315047 [MQTT] MQTT Publish was successful.
798 315048 [MQTT] Notifying task.
799 315048 [OTA Task] Command sent to MQTT task passed.
800 315049 [OTA Task] [OTA] Published 'IN_PROGRESS' status to $aws/things/TI-LaunchPad/
jobs/fe18c7ec_8c31_4438_b0b9_ad55acd9561801 315049 [OTA Task] Sending command to MQTT
 task.
802 315049 [MQTT] Received message d0000 from queue.
803 315150 [MQTT] MQTT Unsubscribe was successful.
804 315150 [MQTT] Notifying task.
805 315151 [OTA Task] Command sent to MQTT task passed.
806 315152 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad/
streams/327

807 315172 [OTA Task] Sending command to MQTT task.
808 315172 [MQTT] Received message e0000 from queue.
809 315273 [MQTT] MQTT Unsubscribe was successful.
810 315273 [MQTT] Notifying task.
811 315274 [OTA Task] Command sent to MQTT task passed.
812 315274 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad/
streams/327

813 315275 [OTA Task] [OTA] Resetting MCU to activate new image.
0 0 [Tmr Svc] Starting Wi-Fi Module ...
1 0 [Tmr Svc] Simple Link task created

Device came up in Station mode

2 137 [Tmr Svc] Wi-Fi module initialized.
```

```
3 137 [Tmr Svc] Starting key provisioning...
4 137 [Tmr Svc] Write root certificate...
5 243 [Tmr Svc] Write device private key...
6 339 [Tmr Svc] Write device certificate...
7 436 [Tmr Svc] Key provisioning done...
Device disconnected from the AP on an ERROR..!!

[WLAN EVENT] STA Connected to the AP: Guest , BSSID: 44:48:c1:ba:b2:c3

[NETAPP EVENT] IP acquired by the device


Device has connected to Guest

Device IP Address is 192.168.3.72


8 1443 [Tmr Svc] Wi-Fi connected to AP Guest.
9 1444 [Tmr Svc] IP Address acquired 192.168.3.72
10 1444 [OTA] OTA demo version 0.9.1
11 1445 [OTA] Creating MQTT Client...
12 1445 [OTA] Connecting to broker...
13 1445 [OTA] Sending command to MQTT task.
14 1445 [MQTT] Received message 10000 from queue.
15 2910 [MQTT] MQTT Connect was accepted. Connection established.
16 2910 [MQTT] Notifying task.
17 2911 [OTA] Command sent to MQTT task passed.
18 2912 [OTA] Connected to broker.
19 2913 [OTA Task] Sending command to MQTT task.
20 2913 [MQTT] Received message 20000 from queue.
21 3014 [MQTT] MQTT Subscribe was accepted. Subscribed.
22 3014 [MQTT] Notifying task.
23 3015 [OTA Task] Command sent to MQTT task passed.
24 3015 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/$next/get/
accepted

25 3028 [OTA Task] Sending command to MQTT task.
26 3028 [MQTT] Received message 30000 from queue.
27 3129 [MQTT] MQTT Subscribe was accepted. Subscribed.
28 3129 [MQTT] Notifying task.
29 3130 [OTA Task] Command sent to MQTT task passed.
30 3138 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/notify-next

31 3138 [OTA Task] [OTA] Check For Update #0
```

```
32 3138 [OTA Task] Sending command to MQTT task.
33 3138 [MQTT] Received message 40000 from queue.
34 3241 [MQTT] MQTT Publish was successful.
35 3241 [MQTT] Notifying task.
36 3243 [OTA Task] Command sent to MQTT task passed.
37 3245 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:TI-LaunchPad ]
38 3245 [OTA Task] [OTA] Set job doc parameter [ jobId:
 fe18c7ec_8c31_4438_b0b9_ad55acd95610 ]
39 3245 [OTA Task] [OTA] Identified job doc parameter [ self_test ]
40 3246 [OTA Task] [OTA] Set job doc parameter [ updatedBy: 589827 ]
41 3246 [OTA Task] [OTA] Set job doc parameter [ streamname: 327 ]
42 3246 [OTA Task] [OTA] Set job doc parameter [ filepath: /sys/mcuflashimg.bin ]
43 3247 [OTA Task] [OTA] Set job doc parameter [ filesize: 130388 ]
44 3247 [OTA Task] [OTA] Set job doc parameter [ fileid: 126 ]
45 3247 [OTA Task] [OTA] Set job doc parameter [ attr: 0 ]
46 3247 [OTA Task] [OTA] Set job doc parameter [ certfile: tisigner.crt.der ]
47 3249 [OTA Task] [OTA] Set job doc parameter [ sig-sha1-rsa:
 Q56qxHRq3Lxv6KkorvilVs4AyGJbWsJd ]
48 3249 [OTA Task] [OTA] Job is ready for self test.
49 3250 [OTA Task] Sending command to MQTT task.
51 3351 [MQTT] MQTT Publish was successful.
52 3352 [MQTT] Notifying task.
53 3352 [OTA Task] Command sent to MQTT task passed.
54 3353 [OTA Task] [OTA] Published 'IN_PROGRESS' status to $aws/things/TI-LaunchPad/
jobs/fe18c7ec_8c31_4438_b0b9_ad55acd95610/u55 3353 [OTA Task] Sending command to MQTT
 task.
56 3353 [MQTT] Received message 60000 from queue.
57 3455 [MQTT] MQTT Unsubscribe was successful.
58 3455 [MQTT] Notifying task.
59 3456 [OTA Task] Command sent to MQTT task passed.
60 3456 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad/streams/327

61 3456 [OTA Task] [OTA] Accepted final image. Commit.
62 3578 [OTA Task] Sending command to MQTT task.
63 3578 [MQTT] Received message 70000 from queue.
64 3779 [MQTT] MQTT Publish was successful.
65 3780 [MQTT] Notifying task.
66 3780 [OTA Task] Command sent to MQTT task passed.
67 3781 [OTA Task] [OTA] Published 'SUCCEEDED' status to $aws/things/TI-LaunchPad/jobs/
fe18c7ec_8c31_4438_b0b9_ad55acd95610/upd68 3781 [OTA Task] [OTA] Returned buffer to
 MQTT Client.
69 4251 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
70 4381 [OTA Task] [OTA] Missing job parameter: execution
71 4382 [OTA Task] [OTA] Missing job parameter: jobId
```

```
72 4382 [OTA Task] [OTA] Missing job parameter: jobDocument
73 4382 [OTA Task] [OTA] Missing job parameter: ts_ota
74 4382 [OTA Task] [OTA] Missing job parameter: files
75 4382 [OTA Task] [OTA] Missing job parameter: streamname
76 4382 [OTA Task] [OTA] Missing job parameter: certfile
77 4382 [OTA Task] [OTA] Missing job parameter: filepath
78 4383 [OTA Task] [OTA] Missing job parameter: filesize
79 4383 [OTA Task] [OTA] Missing job parameter: sig-sha1-rsa
80 4383 [OTA Task] [OTA] Missing job parameter: fileid
81 4383 [OTA Task] [OTA] Missing job parameter: attr
82 4383 [OTA Task] [OTA] Returned buffer to MQTT Client.
83 5251 [OTA] [OTA] Queued: 2   Processed: 2   Dropped: 0
```

Over-the-air configurações de demonstração

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 da aplicação.

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 um jitter do mesmo intervalo.

Baixe, crie, atualize e execute a demonstração do FreeRTOS OTA no Texas Instruments 0SF-LAUNCHXL CC322

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Como fazer download do FreeRTOS e do código de demonstração OTA

- Você pode baixar o código-fonte no GitHub site https://github.com/FreeRTOS/FreeRTOS.

Para criar a aplicação de demonstração

1. Siga as instruções em Conceitos básicos do FreeRTOS para importar o `aws_demos` projeto para o Code Composer Studio, configurar seu AWS IoT endpoint, seu SSID e senha Wi-Fi, além de uma chave privada e certificado para sua placa.
2. Abra *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, comente `#define CONFIG_MQTT_DEMO_ENABLED` e defina `CONFIG_OTA_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 0 [Tmr Svc] Starting Wi-Fi Module ...
1 0 [Tmr Svc] Simple Link task created
Device came up in Station mode
2 142 [Tmr Svc] Wi-Fi module initialized.
3 142 [Tmr Svc] Starting key provisioning...
4 142 [Tmr Svc] Write root certificate...
5 243 [Tmr Svc] Write device private key...
6 340 [Tmr Svc] Write device certificate...
7 433 [Tmr Svc] Key provisioning done...
[WLAN EVENT] STA Connected to the AP: Mobile , BSSID: 24:de:c6:5d:32:a4
[NETAPP EVENT] IP acquired by the device

Device has connected to Mobile
Device IP Address is 192.168.111.12

8 2666 [Tmr Svc] Wi-Fi connected to AP Mobile.
9 2666 [Tmr Svc] IP Address acquired 192.168.111.12
10 2667 [OTA] OTA demo version 0.9.2
11 2667 [OTA] Creating MQTT Client...
12 2667 [OTA] Connecting to broker...
13 3512 [OTA] Connected to broker.
14 3715 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/OtaGA/jobs/
$next/get/accepted
15 4018 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/OtaGA/jobs/
notify-next
16 4027 [OTA Task] [prvPAL_GetPlatformImageState] xFileInfo.Flags = 0250
17 4027 [OTA Task] [prvPAL_GetPlatformImageState] eOTA_PAL_ImageState_Valid
18 4034 [OTA Task] [OTA_CheckForUpdate] Request #0
19 4248 [OTA] [OTA_AgentInit] Ready.
20 4249 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:OtaGA ]
21 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
22 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
23 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
24 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
25 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
26 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: files
27 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
28 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
```

```
29 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
30 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
31 4251 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha1-rsa
32 4251 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
33 4251 [OTA Task] [prvOTA_Close] Context->0x2001b2c4
34 5248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
35 6248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
36 7248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
37 8248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
38 9248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

Baixe, crie, atualize e execute a demonstração do FreeRTOS OTA no Microchip Curiosity MZEF PIC32

Important

Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Como fazer download do código de demonstração OTA do FreeRTOS

- Você pode baixar o código-fonte no GitHub site https://github.com/FreeRTOS/FreeRTOS.

Para criar a aplicação de demonstração de atualização OTA

1. Siga as instruções em Conceitos básicos do FreeRTOS para importar o `aws_demos` projeto para o MPLAB X IDE, configurar seu AWS IoT endpoint, seu SSID e senha Wi-Fi, além de uma chave privada e certificado para sua placa.
2. Abra o `demos/include/aws_ota_codesigner_certificate.h`.
3. Cole o conteúdo do certificado de assinatura de código na variável `static const char signingcredentialSIGNING_CERTIFICATE_PEM`. Seguindo 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_MQTT_DEMO_ENABLED` e defina `CONFIG_OTA_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 sua placa e execute o projeto na sua placa para confirmar se ele pode se conectar ao Wi-Fi e ao agente de mensagens MQTT. AWS IoT

Quando você executa o projeto, o MPLAB X IDE deve abrir uma janela de saída. Verifique se a ICD4guia está selecionada. Você verá 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
```

Esta saída mostra que o Microchip Curiosity PIC32 MZEF pode se conectar AWS IoT e assinar os tópicos do MQTT necessários para atualizações do OTA. As mensagens `Missing job parameter` são esperadas porque não há tarefas de atualização OTA pendentes.

Baixe, crie, atualize e execute a demonstração do FreeRTOS OTA no Espressif ESP32

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

1. Faça o download da fonte dos FreeRTOS em. 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_MQTT_DEMO_ENABLED` e defina `CONFIG_OTA_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_MQTT_DEMO_ENABLED` e defina `CONFIG_OTA_UPDATE_DEMO_ENABLED`.
   - Certifique-se de que seu certificado de assinatura de código SHA-256/ECDSA seja copiado no `demos/include/aws_ota_codesigner_certificate.h`.

Baixe, crie, atualize e execute a demonstração do FreeRTOS OTA no Renesas N RX65

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este capítulo mostra como baixar, criar, atualizar e executar os aplicativos de demonstração FreeRTOS OTA no Renesas N. RX65

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: RSKRX65 N-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 RX65 N e ao PC.
2. Conecte a fonte de alimentação ao RX65 N.

Configure seus AWS recursos

1. Para executar as demonstrações do FreeRTOS, você deve ter AWS uma conta com AWS Identity and Access Management um usuário (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.

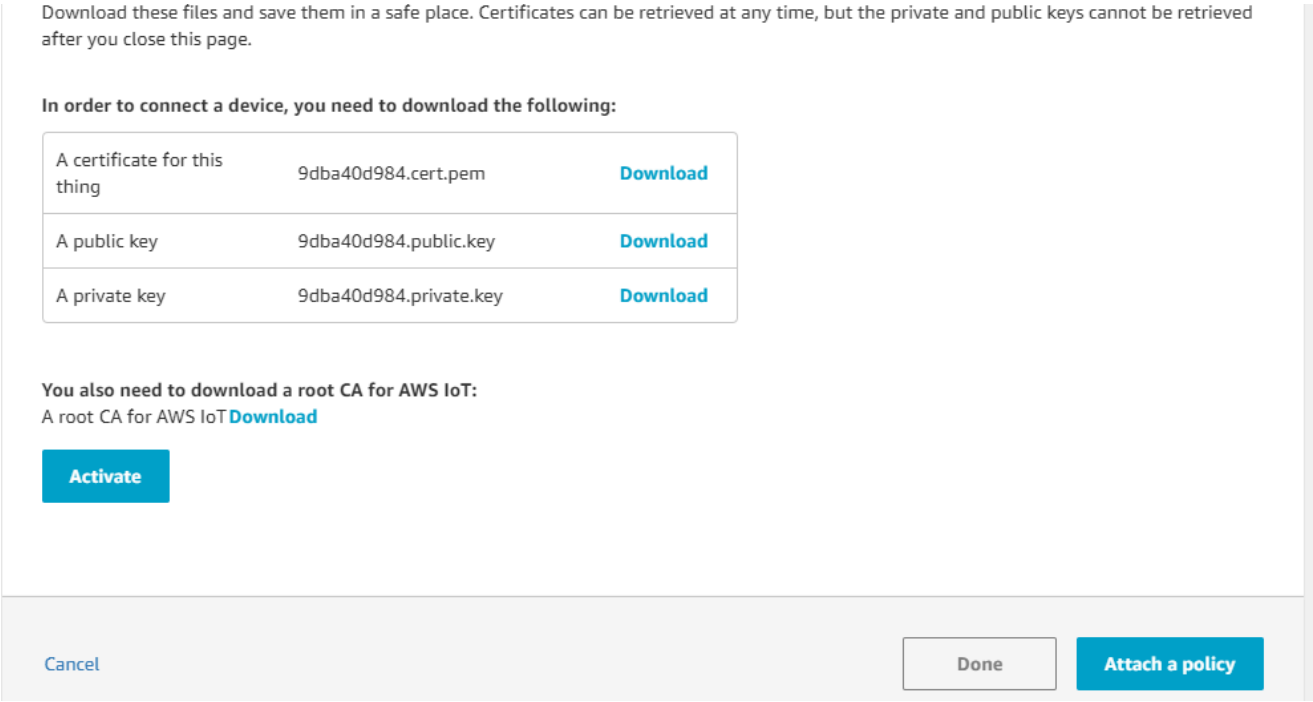
a. Escolha Criar e Criar uma única coisa.

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 “Gerando pares de SHA256 chaves ECDSA com OpenSSL” na Política de design de atualização de firmware da Renesas MCU.

   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.

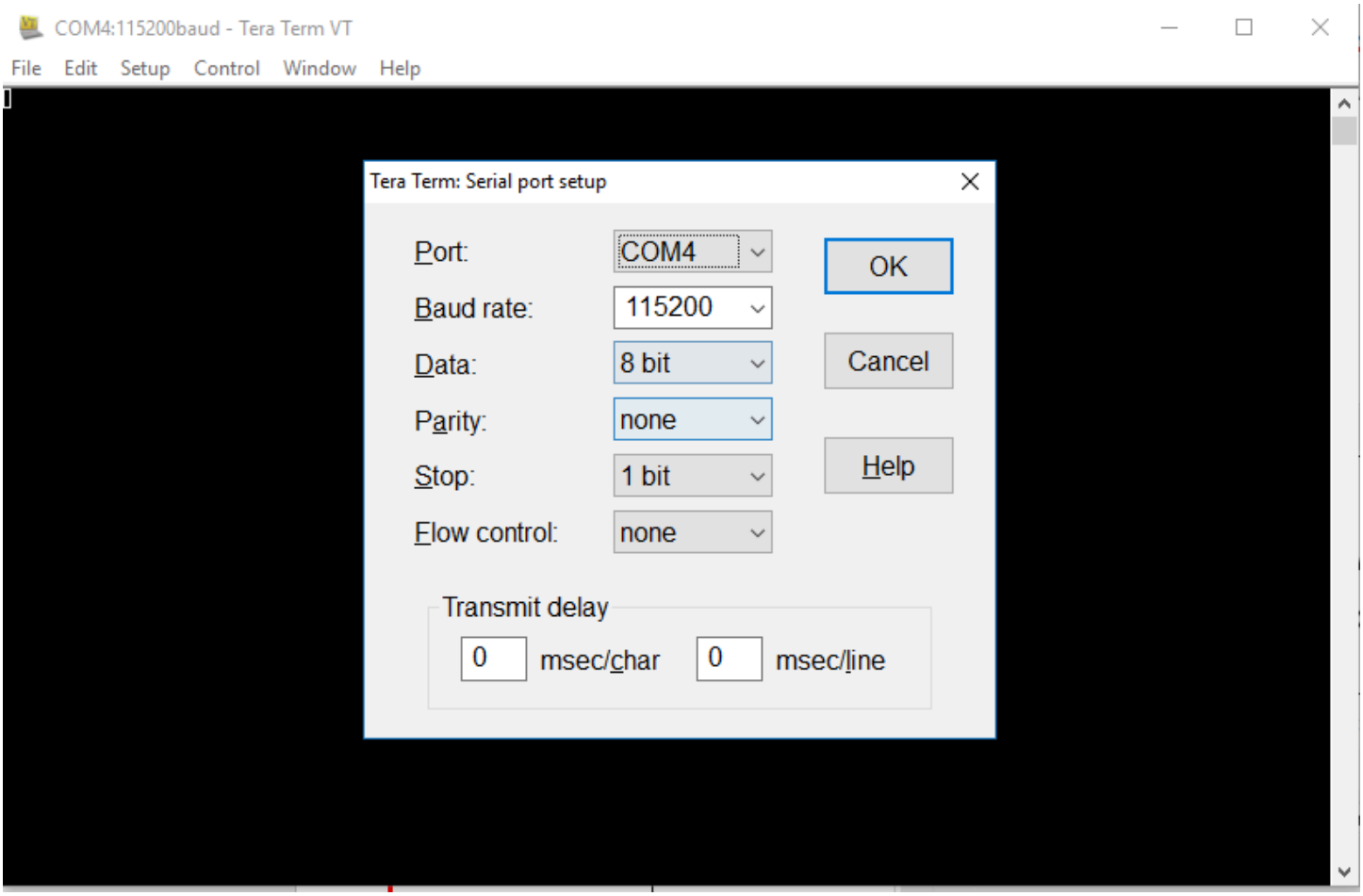
      iv. Na janela Criar um perfil de assinatura de código, insira um nome de perfil. Para a plataforma de hardware de dispositivo, selecione o Windows Simulator. Para o certificado de assinatura de código, escolha Importar.

      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. Escolha Criar.

6. Para conceder acesso à assinatura de código para AWS IoT, siga as etapas emConceda acesso à assinatura de código para AWS IoT.

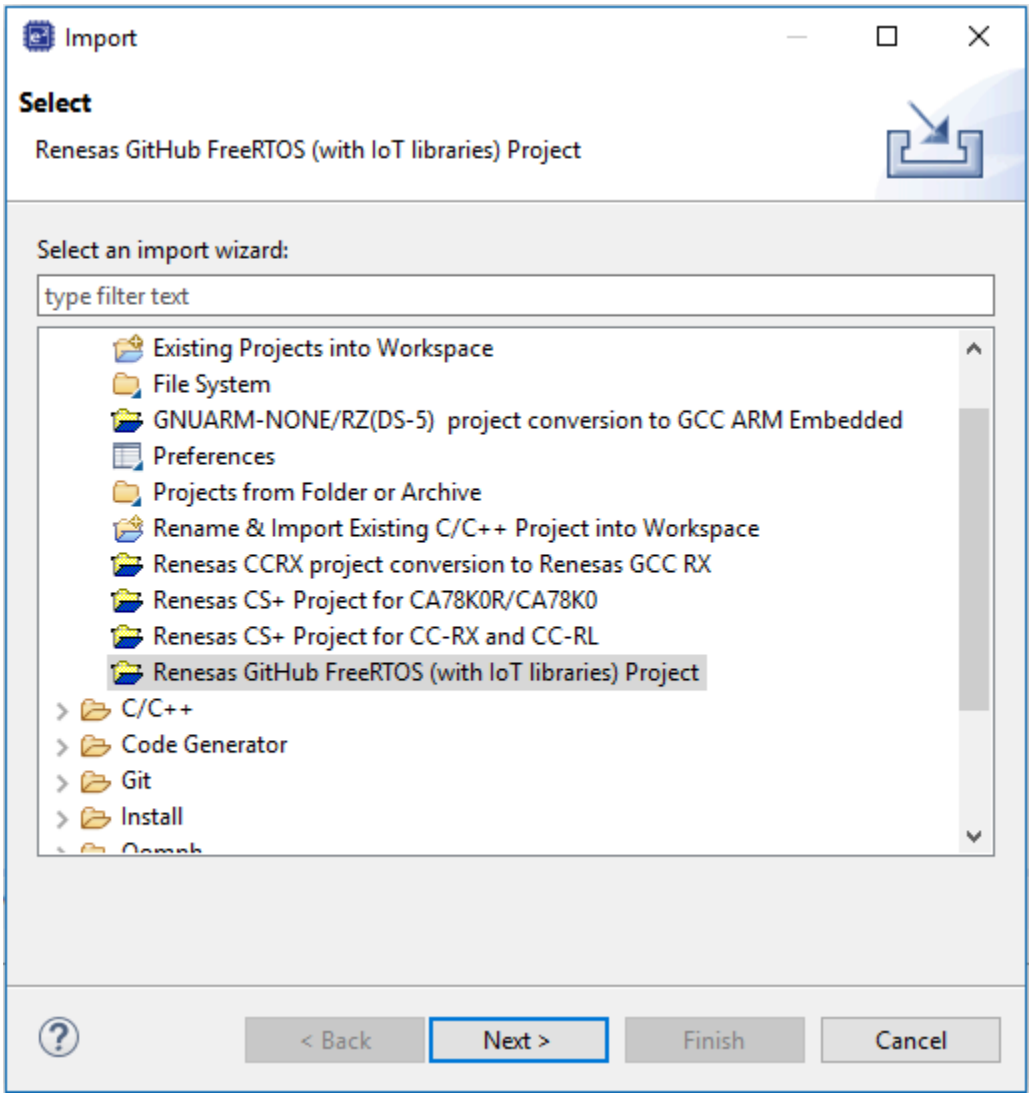
Se não tiver o Tera Term instalado em seu PC, você pode baixá-lo https://ttssh2.osdn.jp/index.htmlem .en e configurá-lo conforme mostrado aqui. Conecte a porta serial USB do dispositivo ao PC.



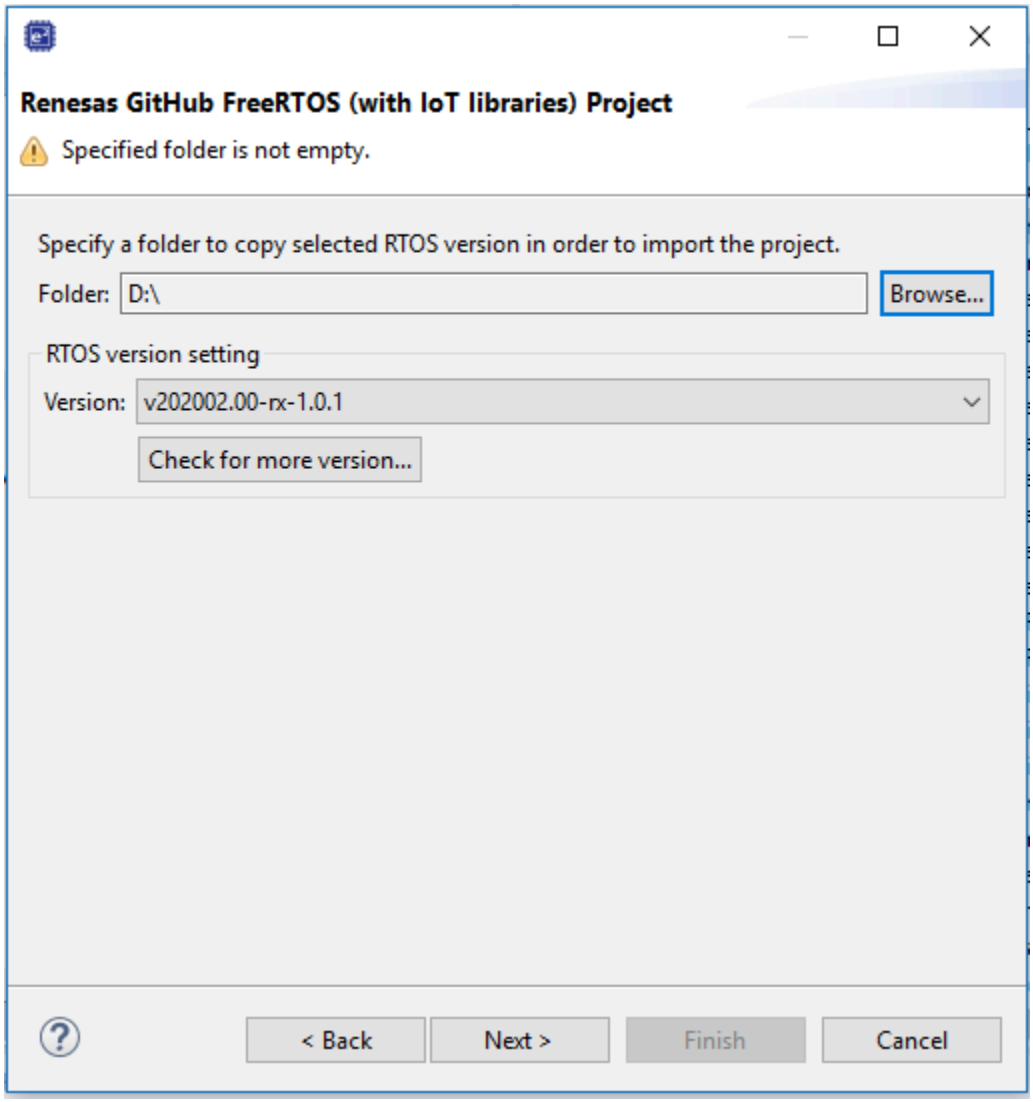
Importe, configure o arquivo de cabeçalho e compile aws_demos e boot_loader

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.

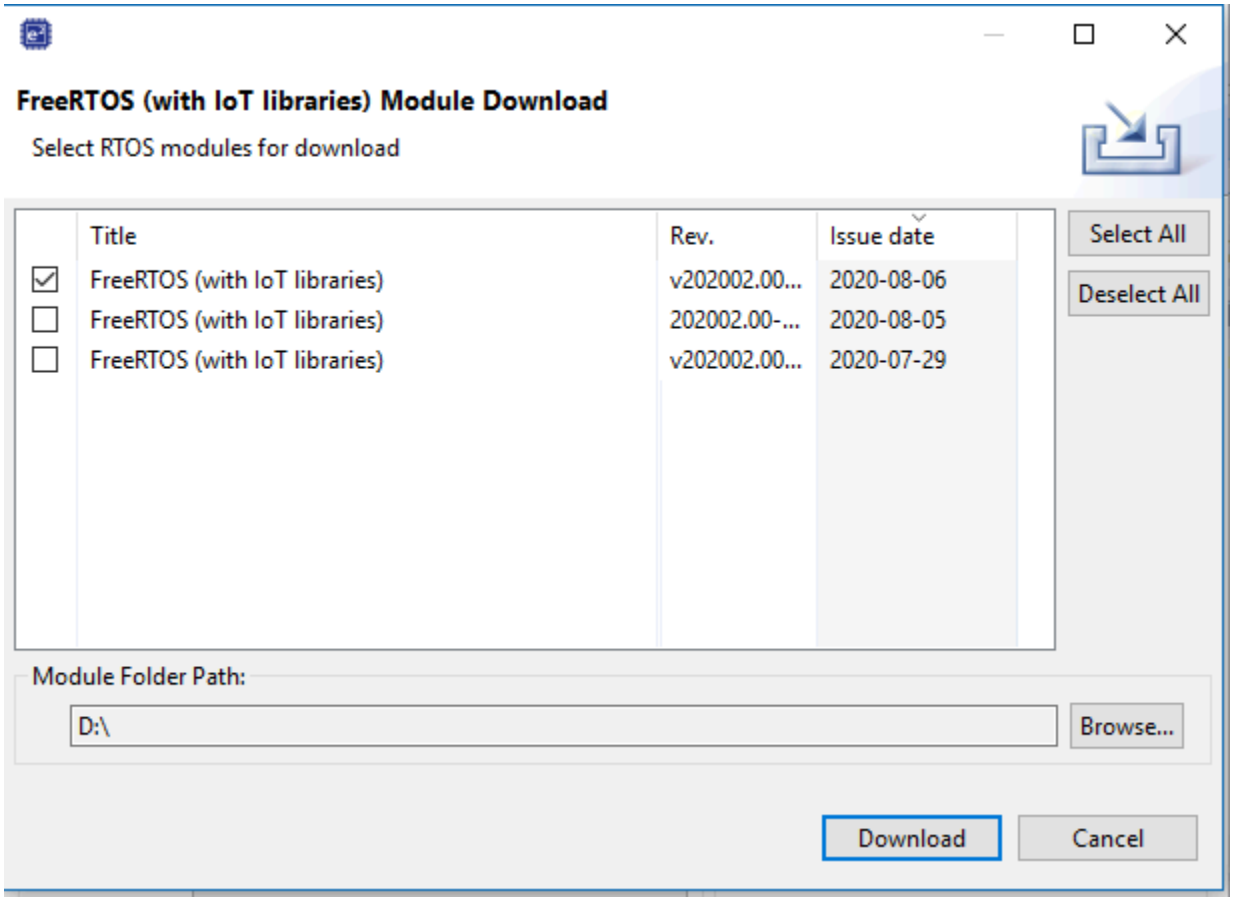
1. Lançamento do e^2 studio.
2. Escolha o Arquivo e depois Importar....
3. Selecione o projeto Renesas GitHub FreeRTOS (com bibliotecas de IoT).



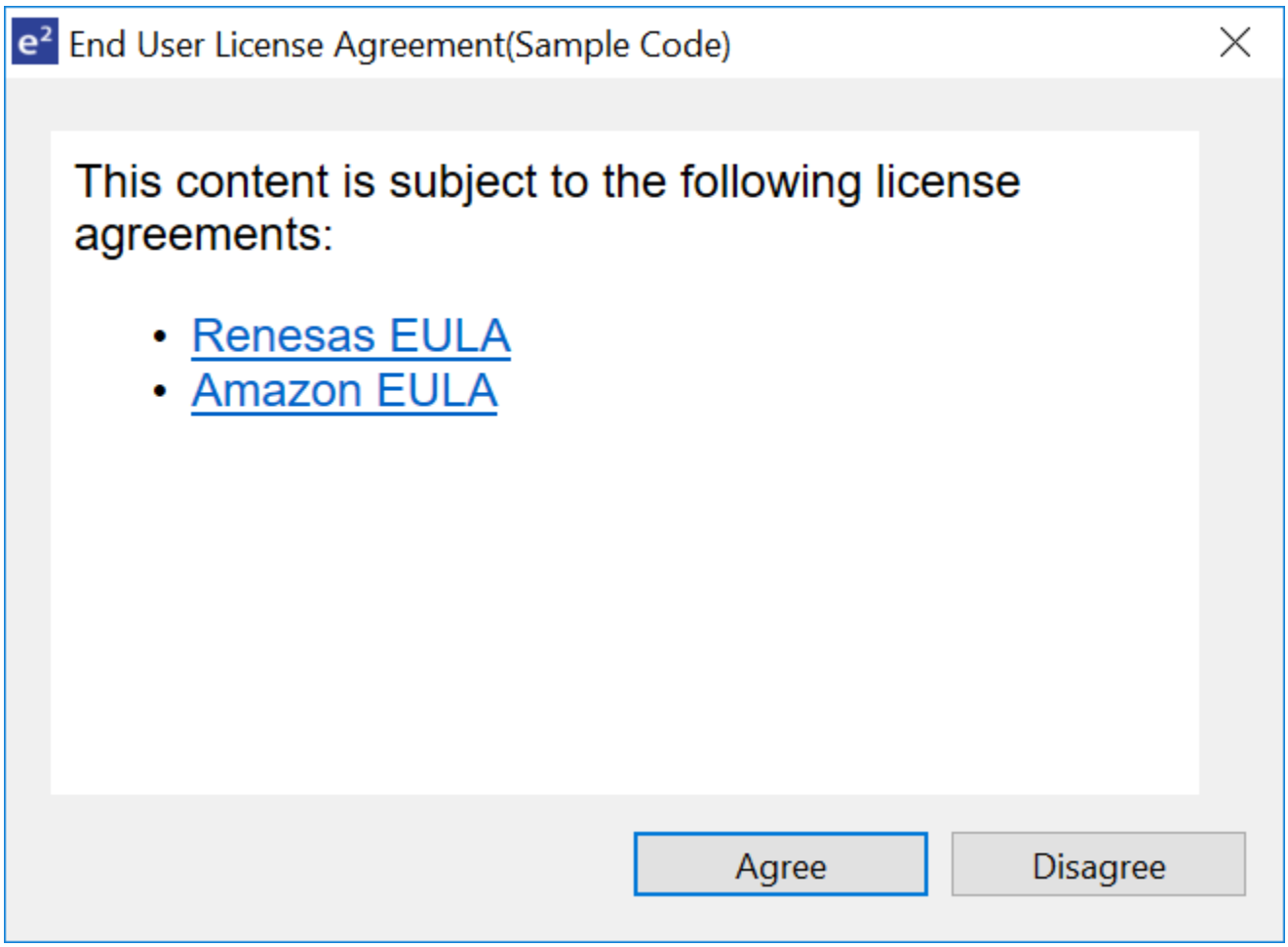
4. Escolha Verificar mais versões... para mostrar a caixa de diálogo de download.



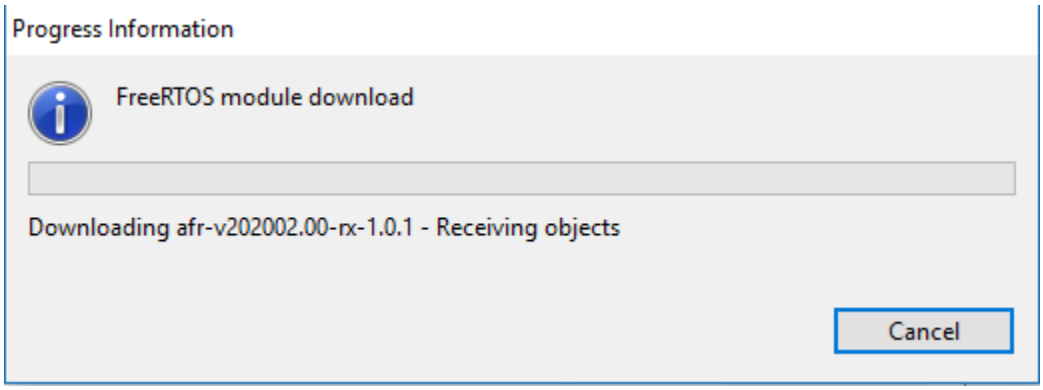
5. Selecione o pacote mais recente.



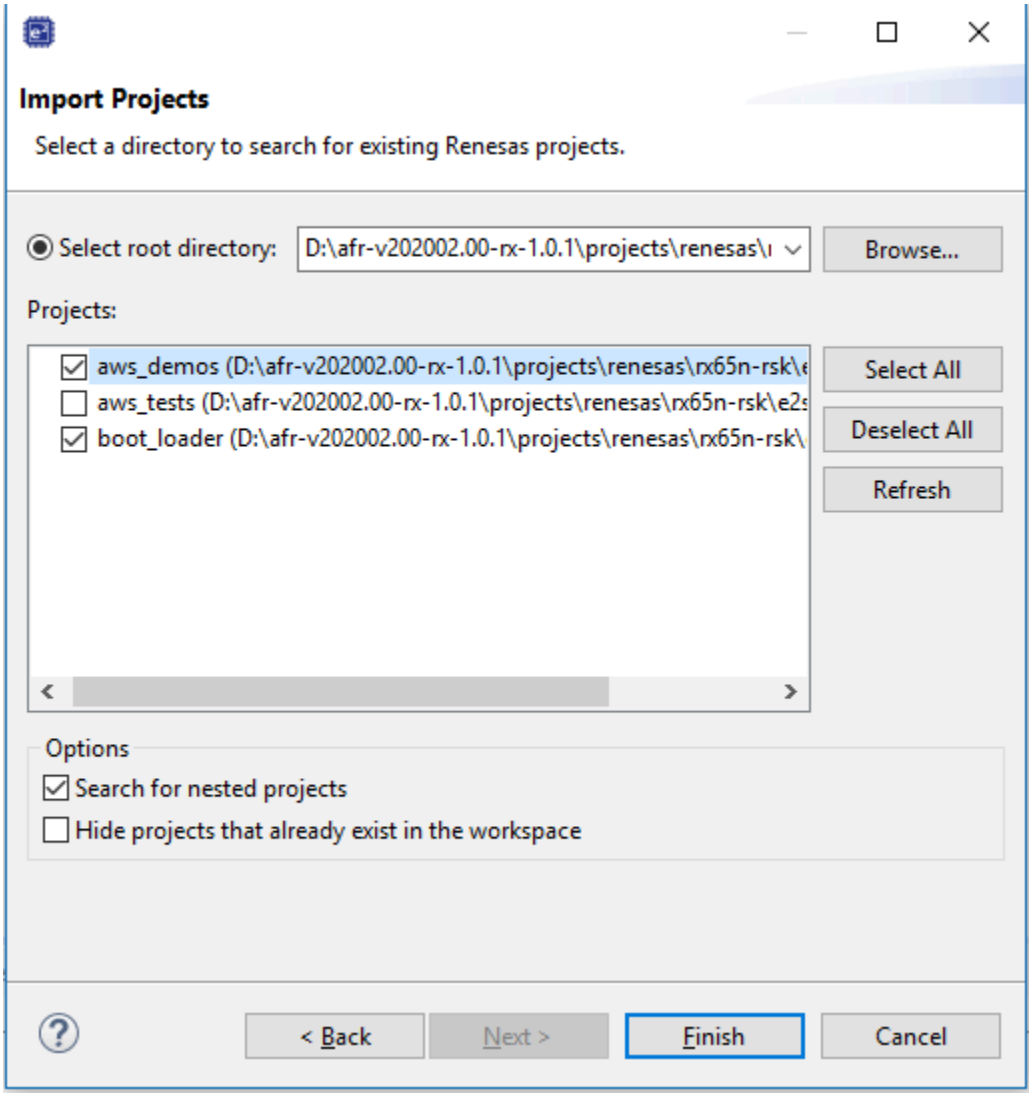
6. Escolha Concordo para aceitar o contrato de licença do usuário final.



7. Aguarde a conclusão do download.



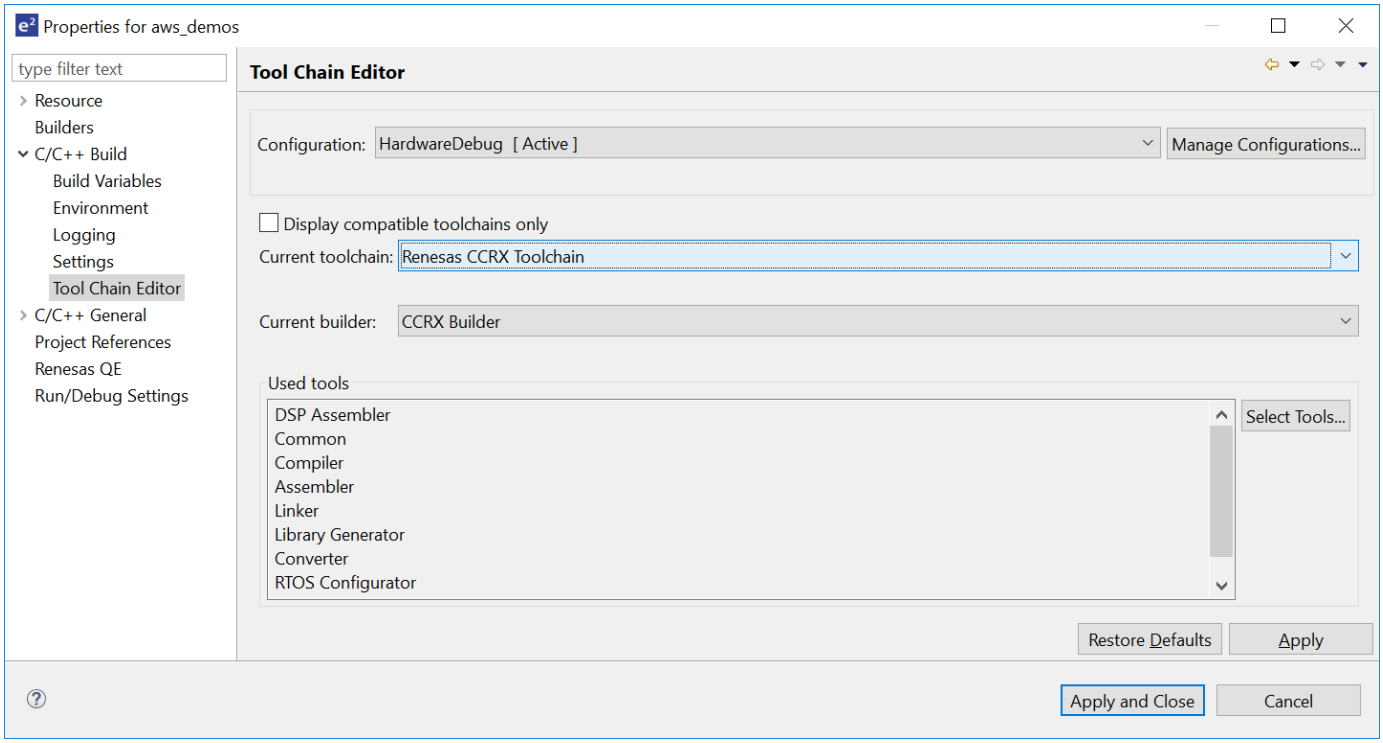
8. Selecione os projetos aws_demos e boot_loader e escolha Concluir para importá-los.



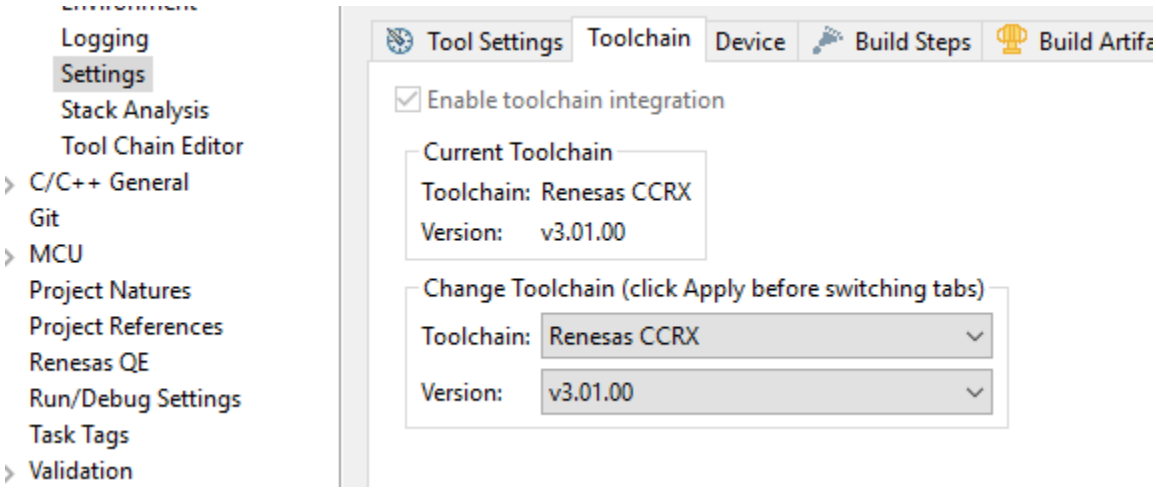
9. Para ambos os projetos, abra as propriedades do projeto. No painel de navegação, selecione Editor de cadeia de ferramentas.

    a. Escolha a Cadeia de ferramentas atual.

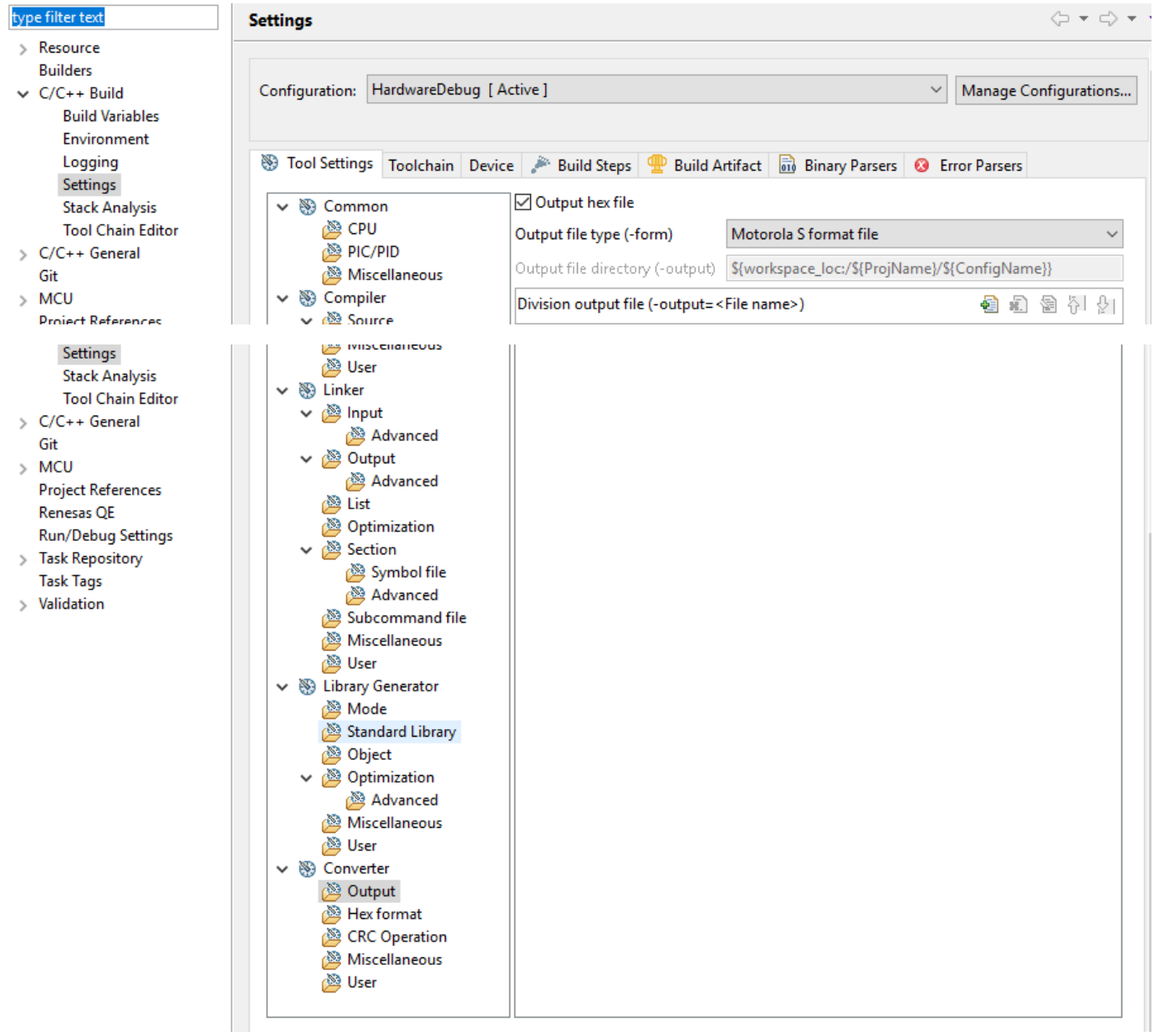
    b. Escolha o Builder atual.



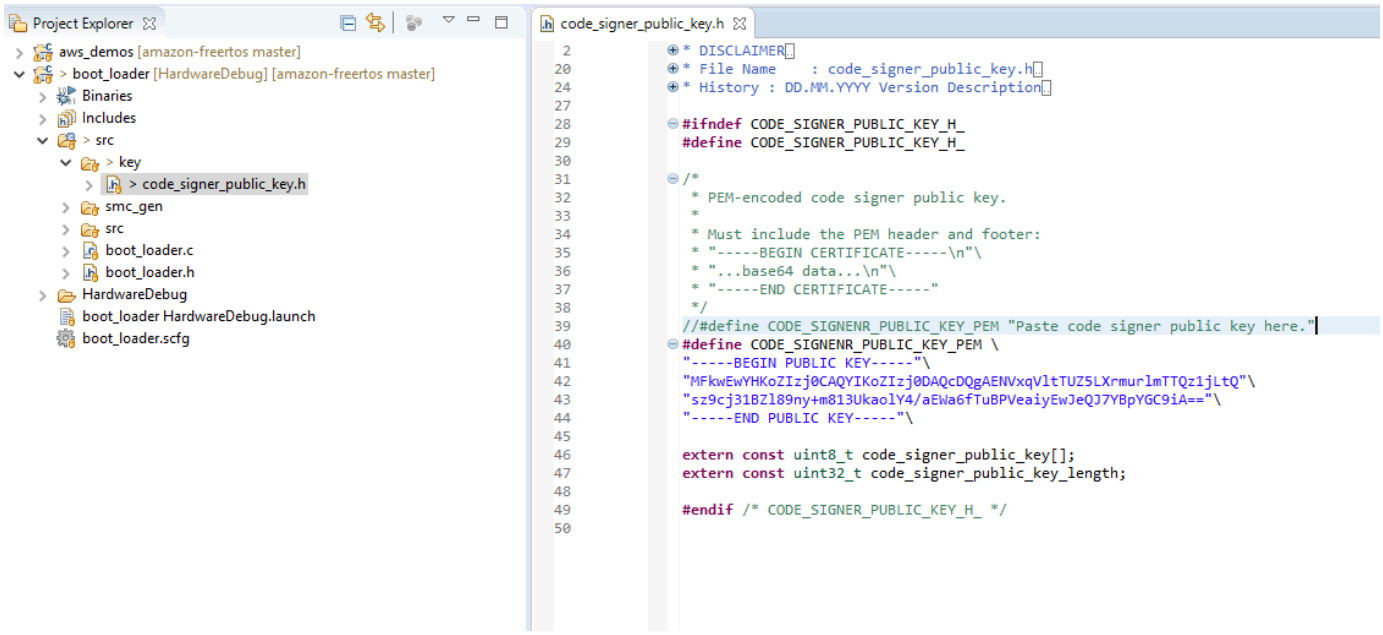
10. No painel de navegação, selecione Configurações. Escolha a guia Cadeia de ferramentas e, em seguida, escolha a Versão da cadeia de ferramentas.



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 FreeRTOS OTA usando o Amazon Web Services RX65 em N e a seção 7.3 “Gerando pares de chaves ECDSA com OpenSSL” na Política de SHA256 design de atualização de firmware da Renesas MCU.



Em seguida, compile o projeto para criar `boot_loader.mot`.

12. Abra o projeto `aws_demos`.

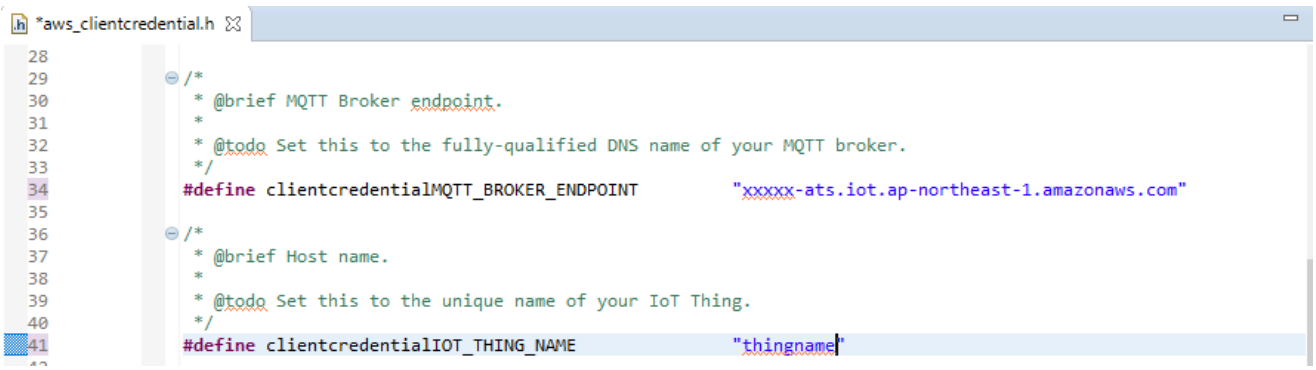
    a. Abra o console de AWS IoT.

    b. No painel de navegação à esquerda, escolha Configurações. Anote o endpoint personalizado.

    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.

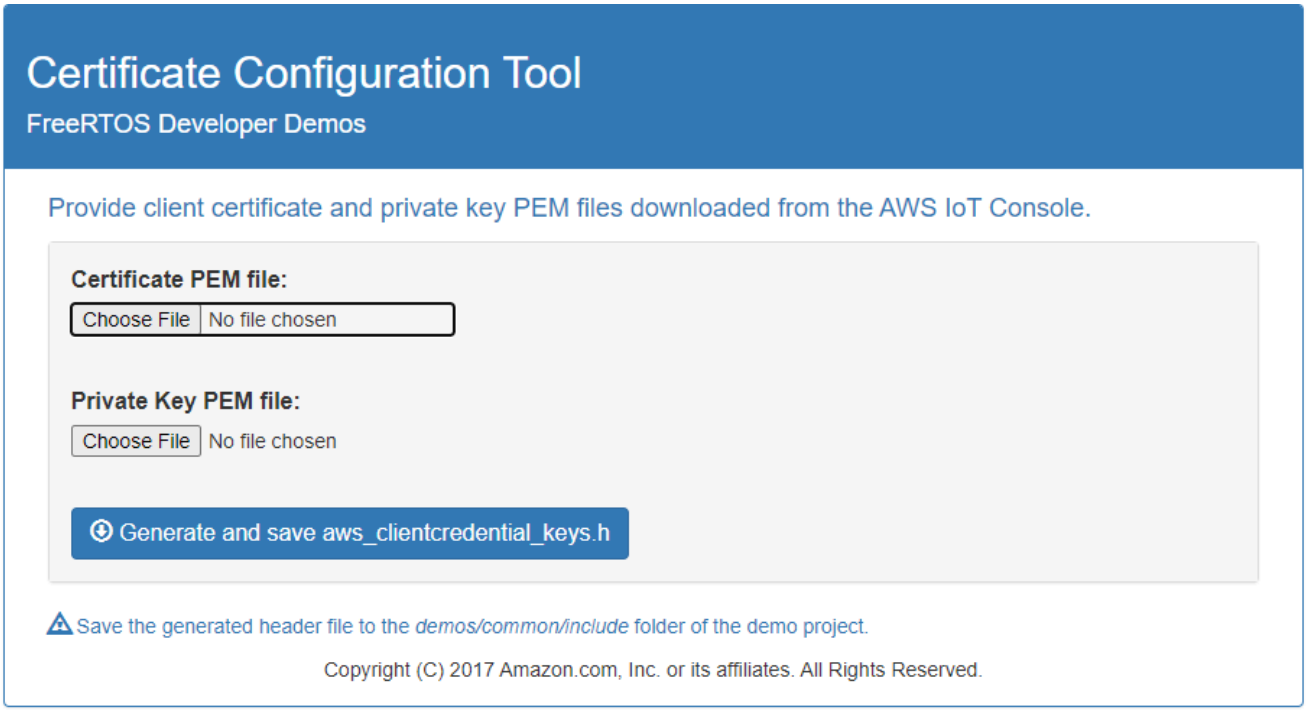
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



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.

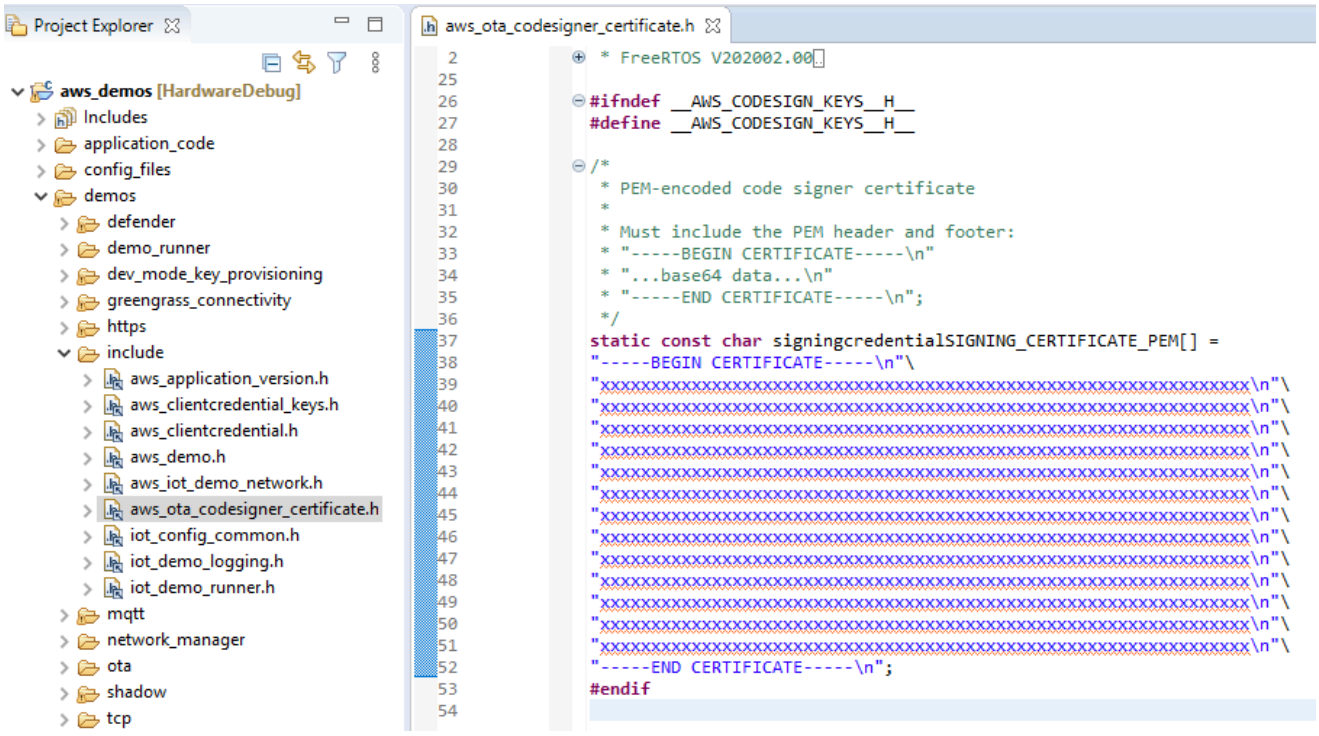
g. Escolha Gerar e salvar aws_clientcredential_keys.h e substitua esse arquivo no diretório `demos/include/`.



h. Abra o arquivo `demos/include/aws_ota_codesigner_certificate.h` e especifique esses valores.

```
#define signingcredentialSIGNING_CERTIFICATE_PEM [] = "your-certificate-key";
```

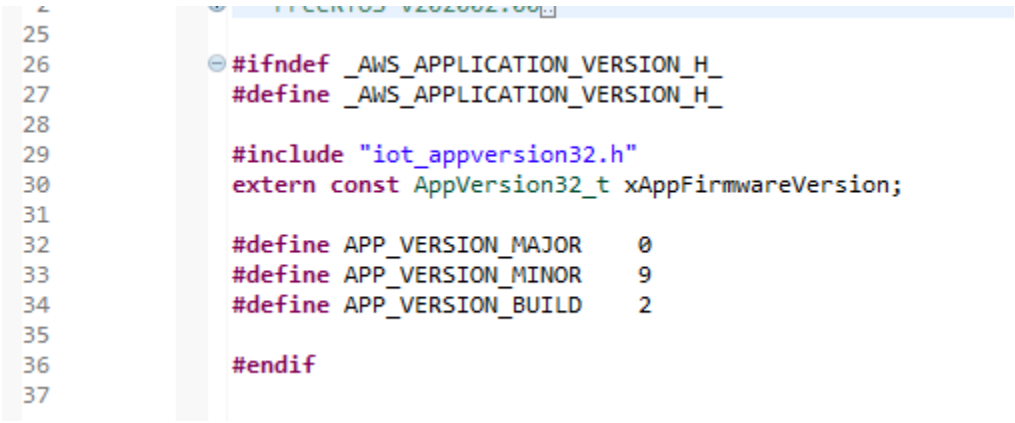
Onde *`your-certificate-key`* está o valor do arquivo`secp256r1.crt`. Lembre-se de adicionar "\" após cada linha na certificação. Para obter mais informações sobre a criação do `secp256r1.crt` arquivo, consulte Como implementar o FreeRTOS OTA usando o Amazon Web Services RX65 em N e a seção 7.3 “Gerando pares de chaves ECDSA com OpenSSL” na Política de SHA256 design de atualização de firmware da Renesas MCU.



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_MQTT_DEMO_ENABLED` e defina `CONFIG_OTA_UPDATE_DEMO_ENABLED`.

    b. Abra o arquivo `demos/include/ aws_application_version.h` e defina a versão inicial do firmware como `0.9.2`.



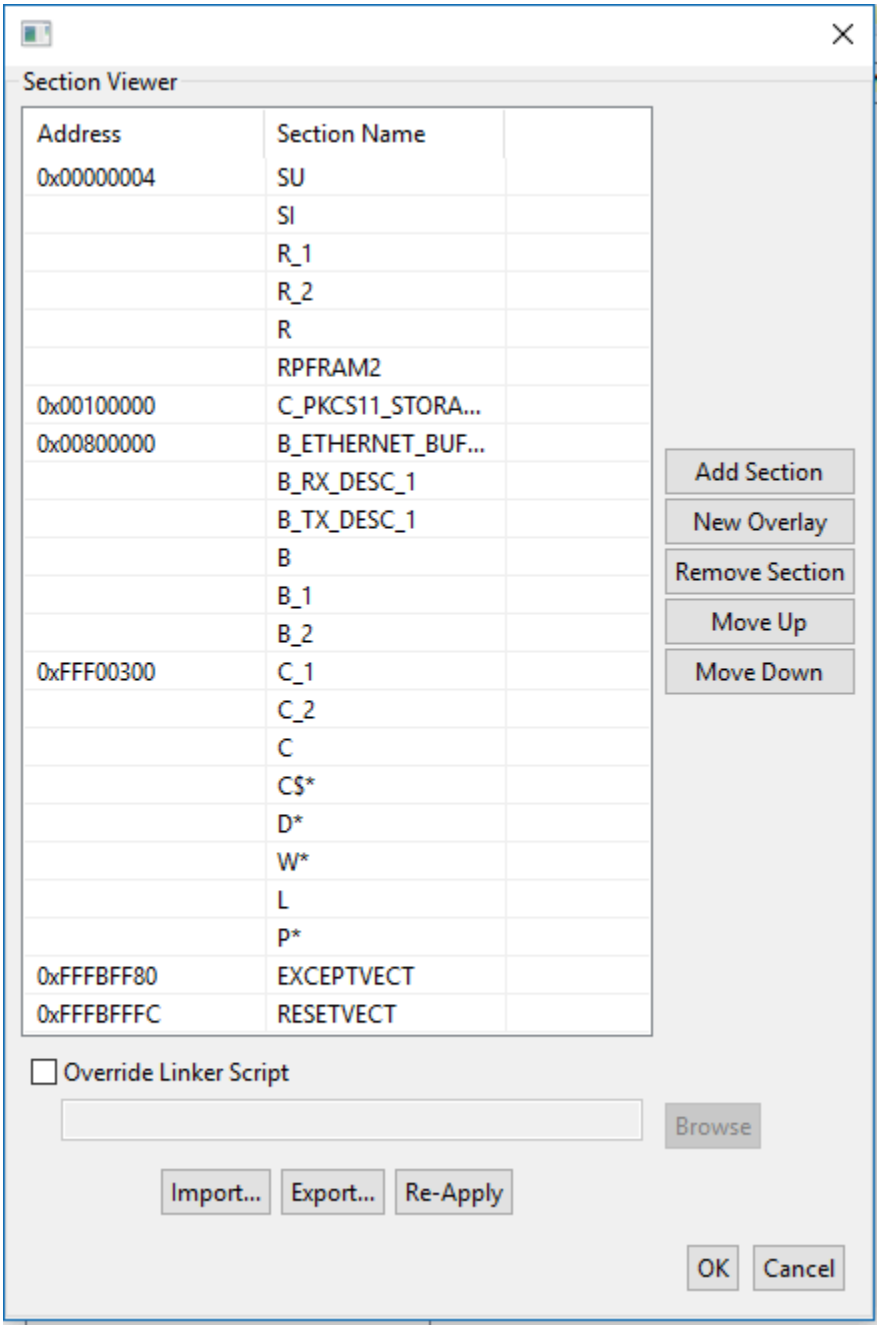
```
#ifndef _AWS_APPLICATION_VERSION_H_
#define _AWS_APPLICATION_VERSION_H_

#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.



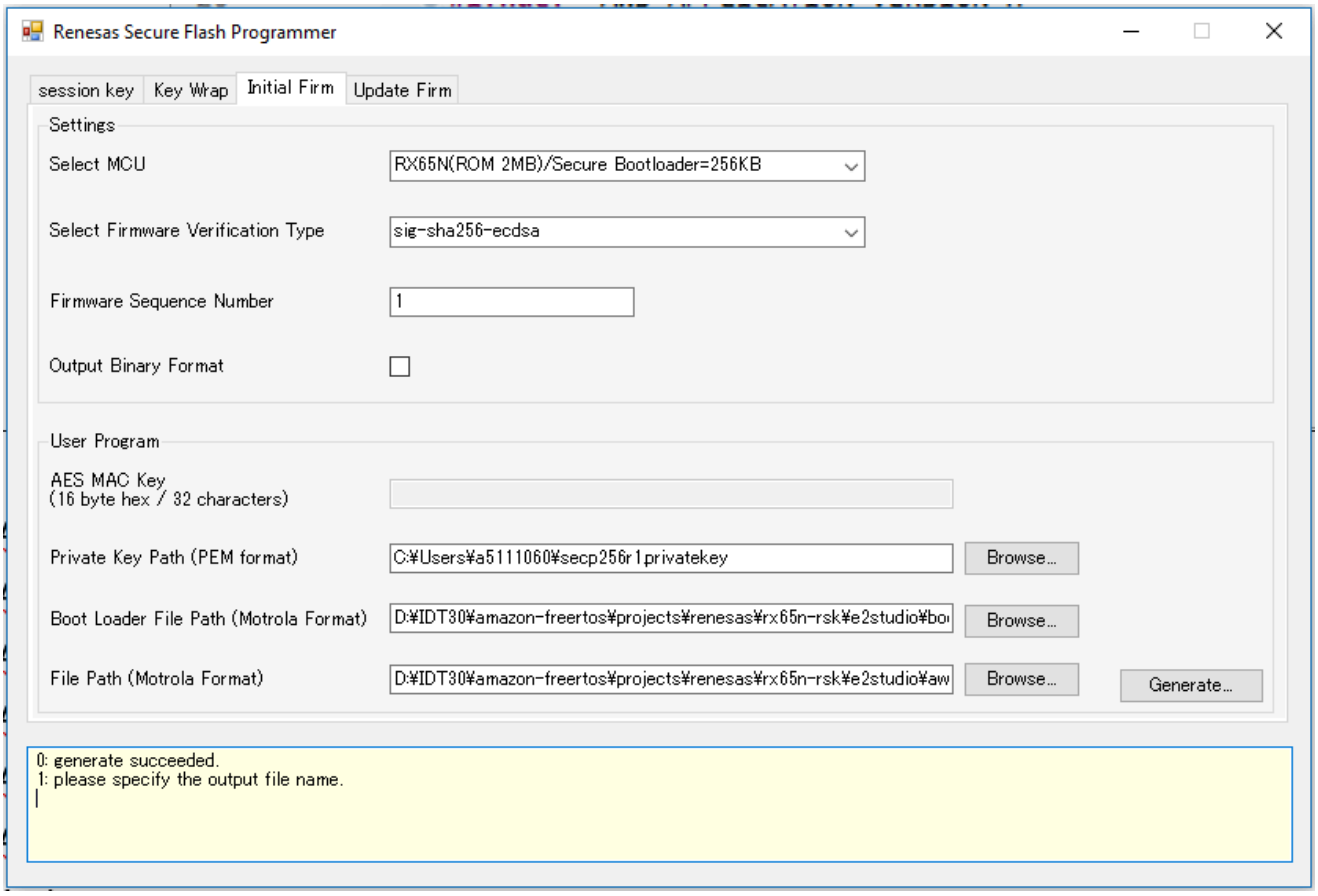
d. Escolha Compilar para criar o arquivo `aws_demos.mot`.

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 RX65 N-RSK para instalar o firmware inicial.

    a. Baixe o 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`).



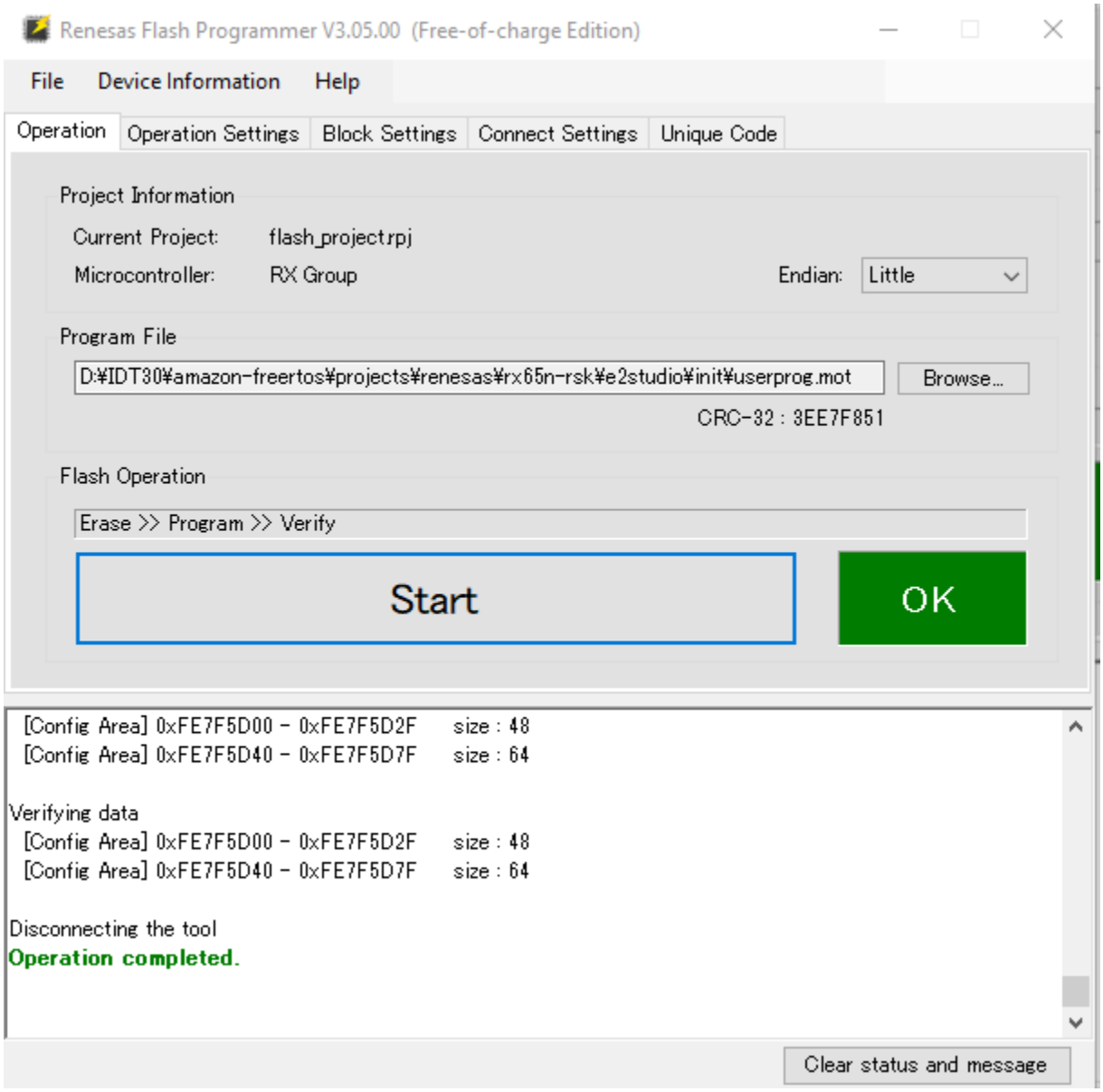
c. Crie um diretório chamado `init_firmware` gere `userprog.mot` e salve-o no diretório `init_firmware`. Verifique se o que foi gerado teve êxito.

15. Atualize o firmware inicial no RX65 N-RSK.

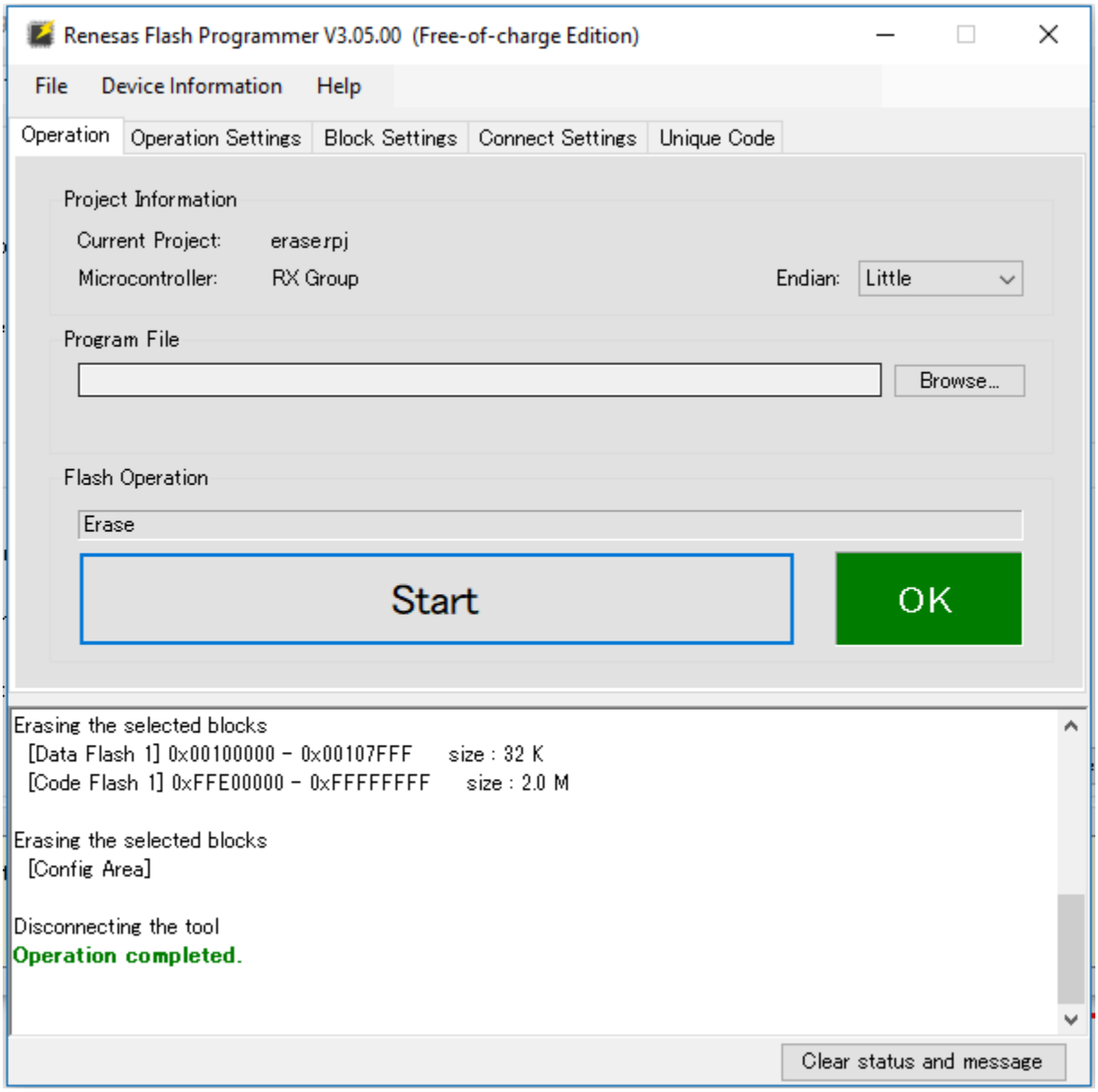
a. [Baixe a versão mais recente do Renesas Flash Programmer (GUI de programação) em - .html. https://www.renesas.com/tw/ en/products/software-tools/tools/programmer/renesas flash-programmer-programming-gui]

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 em seu RX65 N-RSK. A placa RX65 N-RSK agora está ouvindo as atualizações do 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.

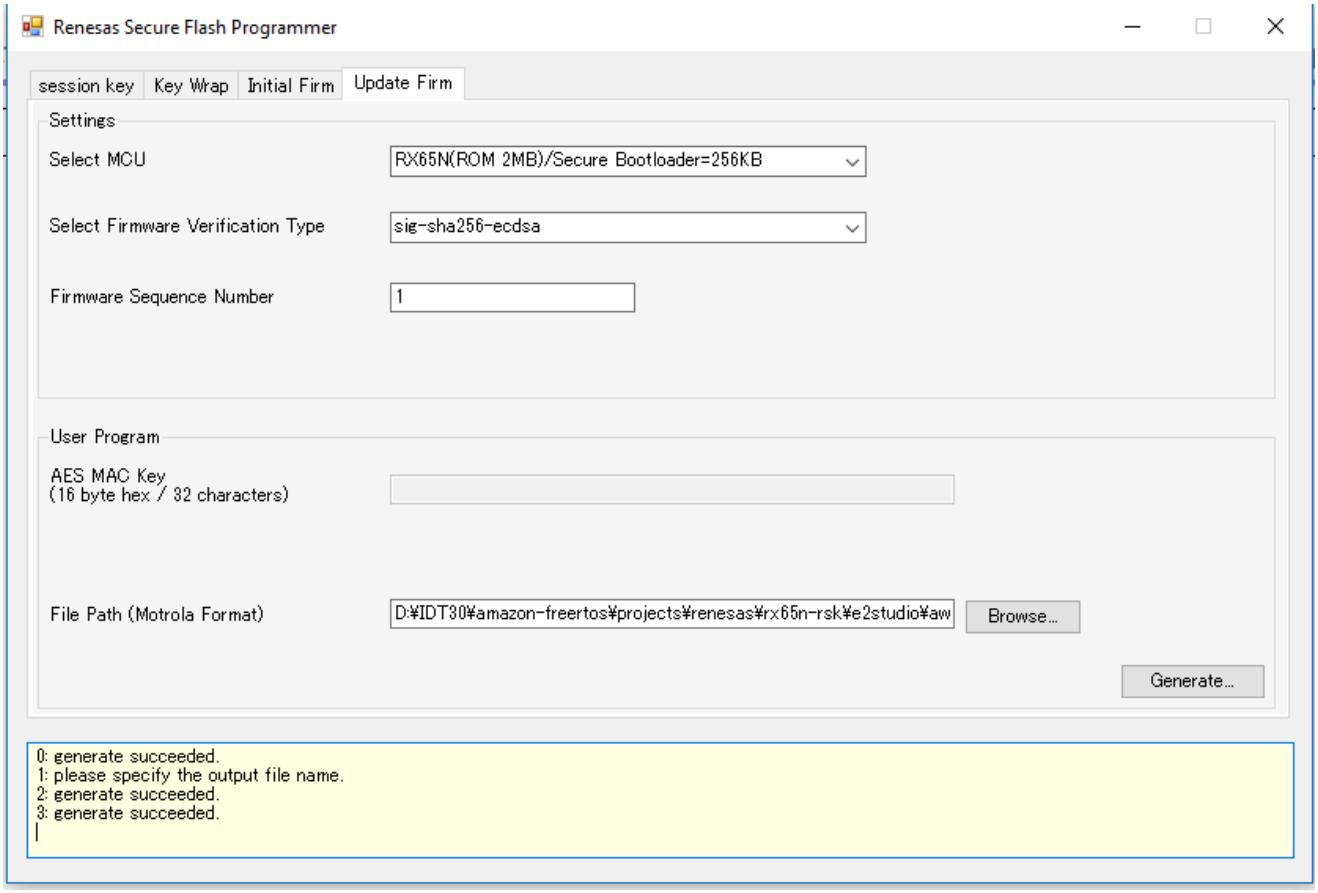
18. Crie o arquivo `userprog.rsu` com o Renesas Secure Flash Programmer para atualizar a versão do seu firmware.

    a. Abra o arquivo `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe`.

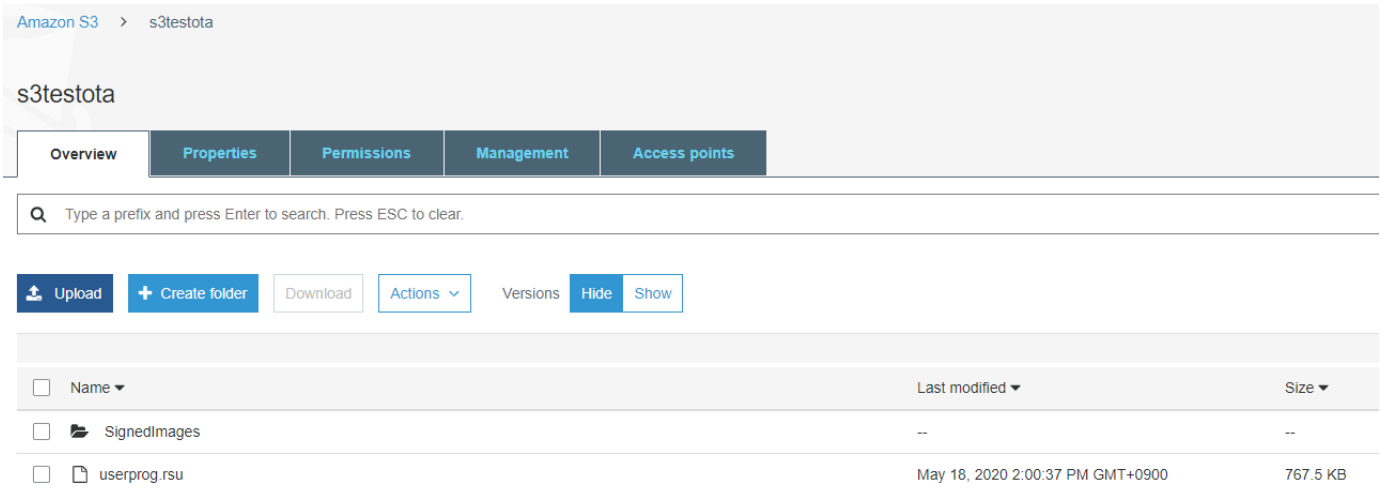
    b. Escolha a guia Atualizar firmware e defina os seguintes parâmetros:

    - 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 tarefa para atualizar o firmware no RX65 N-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.

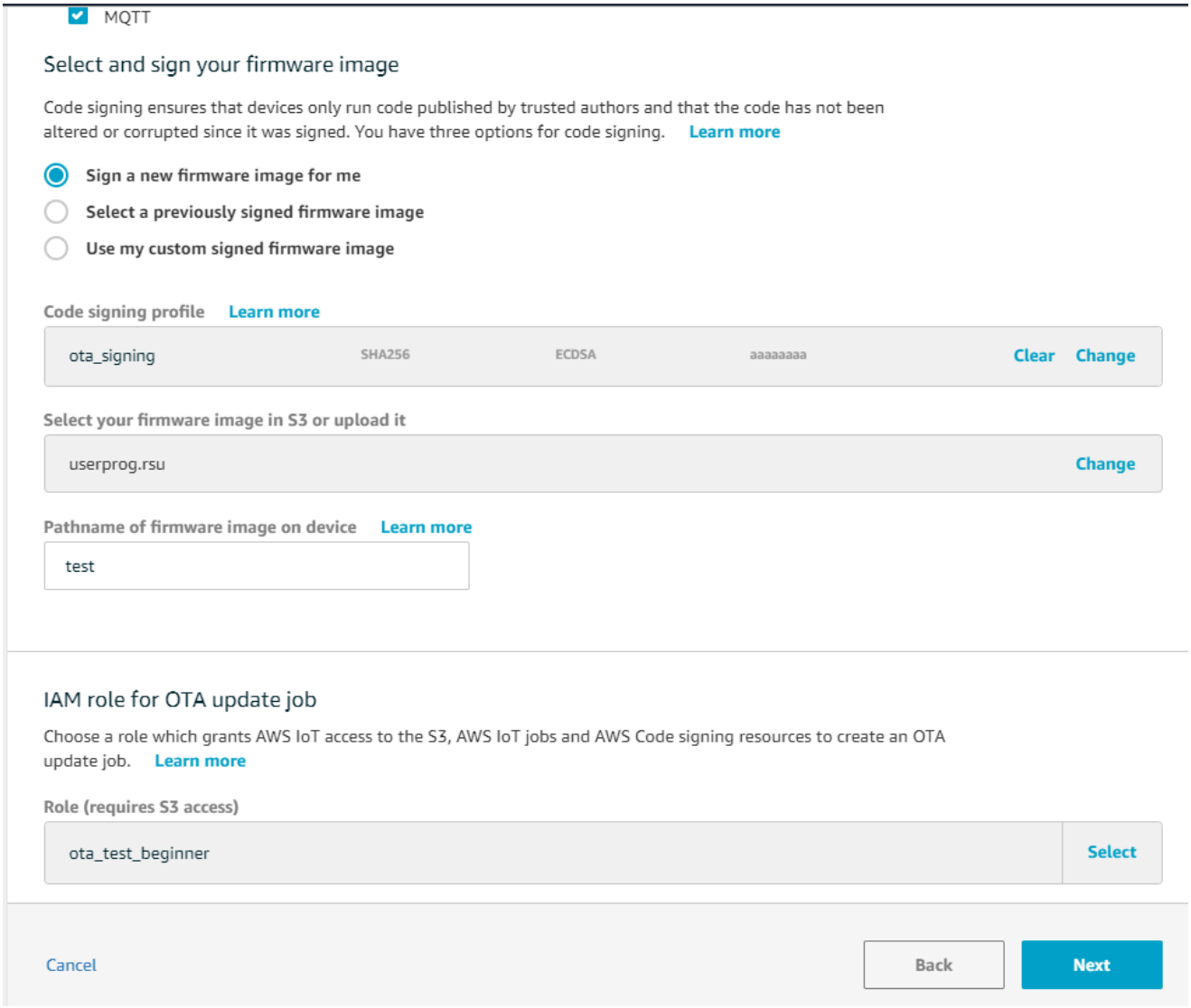
    a. Faça login no console do AWS IoT. No painel de navegação, escolha Gerenciar e escolha Trabalhos.

    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:

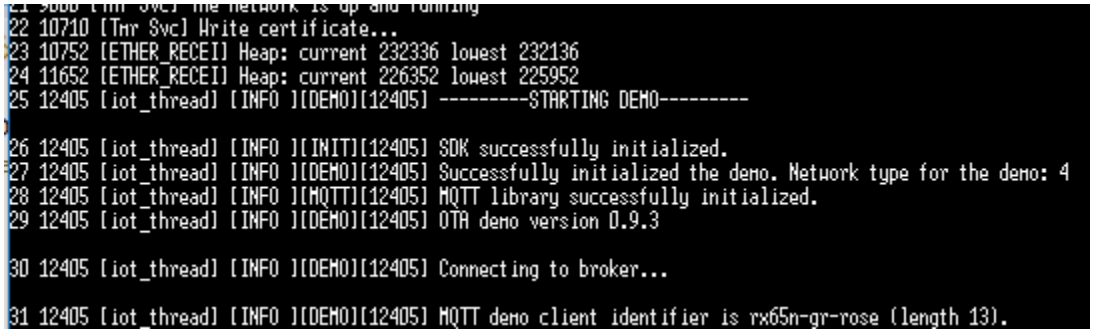
    - Escolha MQTT.
    - Selecione o perfil de assinatura de código criado na seção anterior.
    - Selecione a imagem do firmware que você carregou em um bucket do Amazon S3.
    - Em Nome do caminho da imagem do firmware no dispositivo, insira **`test`**.
    - Selecione o perfil do IAM que você criou na seção anterior.

    d. Escolha Próximo.

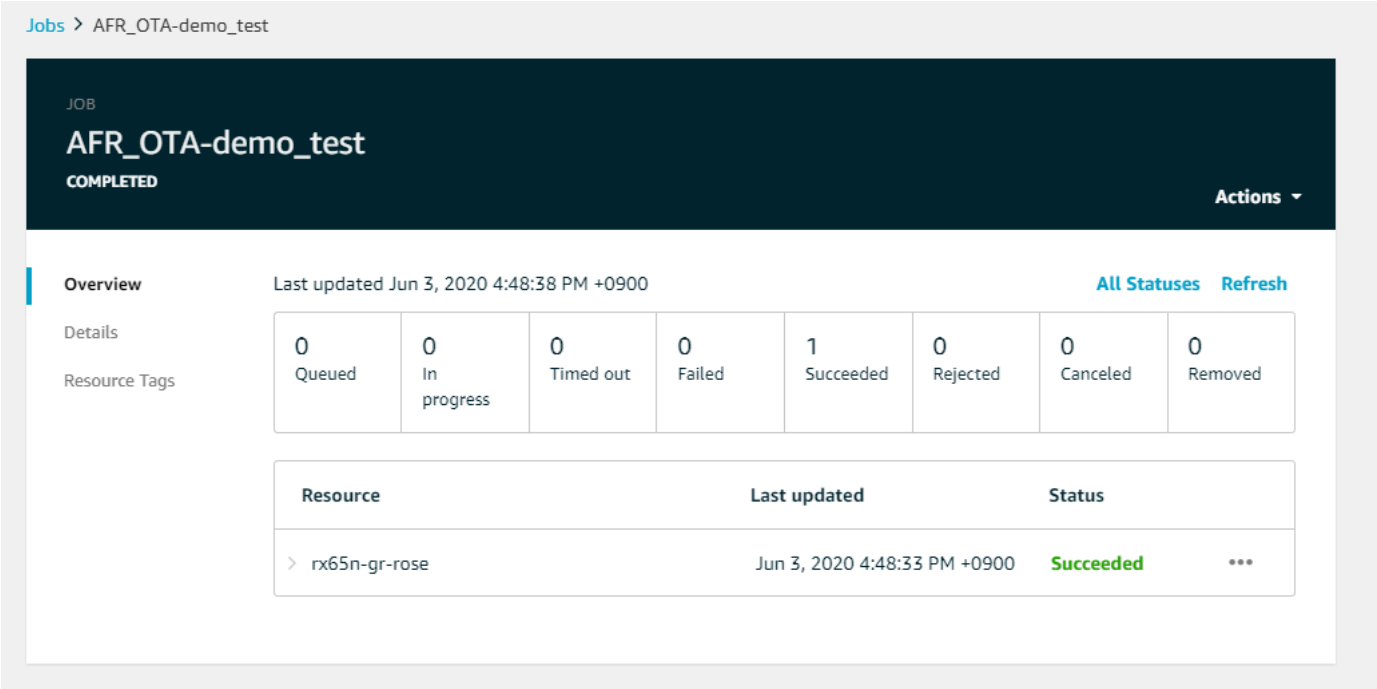


e. Insira um ID e escolha Criar.

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.



AWS IoT Aplicativo de demonstração Device Shadow

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Introdução

Esta demonstração mostra como usar a biblioteca AWS IoT Device Shadow para se conectar ao serviço AWS Device Shadow. Ele usa o Biblioteca coreMQTT para estabelecer uma conexão MQTT com TLS (Autenticação Mútua) com o AWS IoT MQTT Broker e o analisador da biblioteca CoreJSON para analisar documentos paralelos recebidos do serviço Shadow. 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 pela mesma aplicação. Em um cenário de produção realista, uma aplicação externa 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.

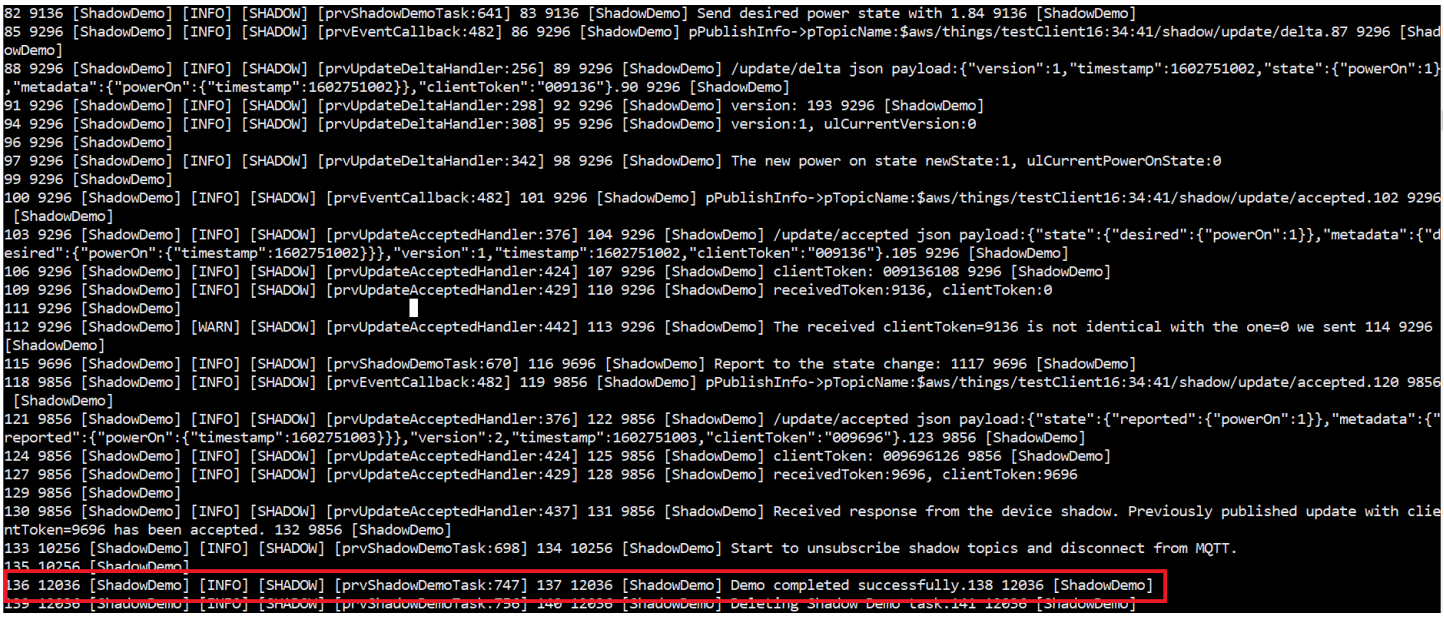
Funcionalidade

A demonstração cria uma tarefa de aplicação 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 AWS IoT MQTT Broker e presume que há um `power0n` estado na sombra do dispositivo.

A demonstração realiza as seguintes operações:

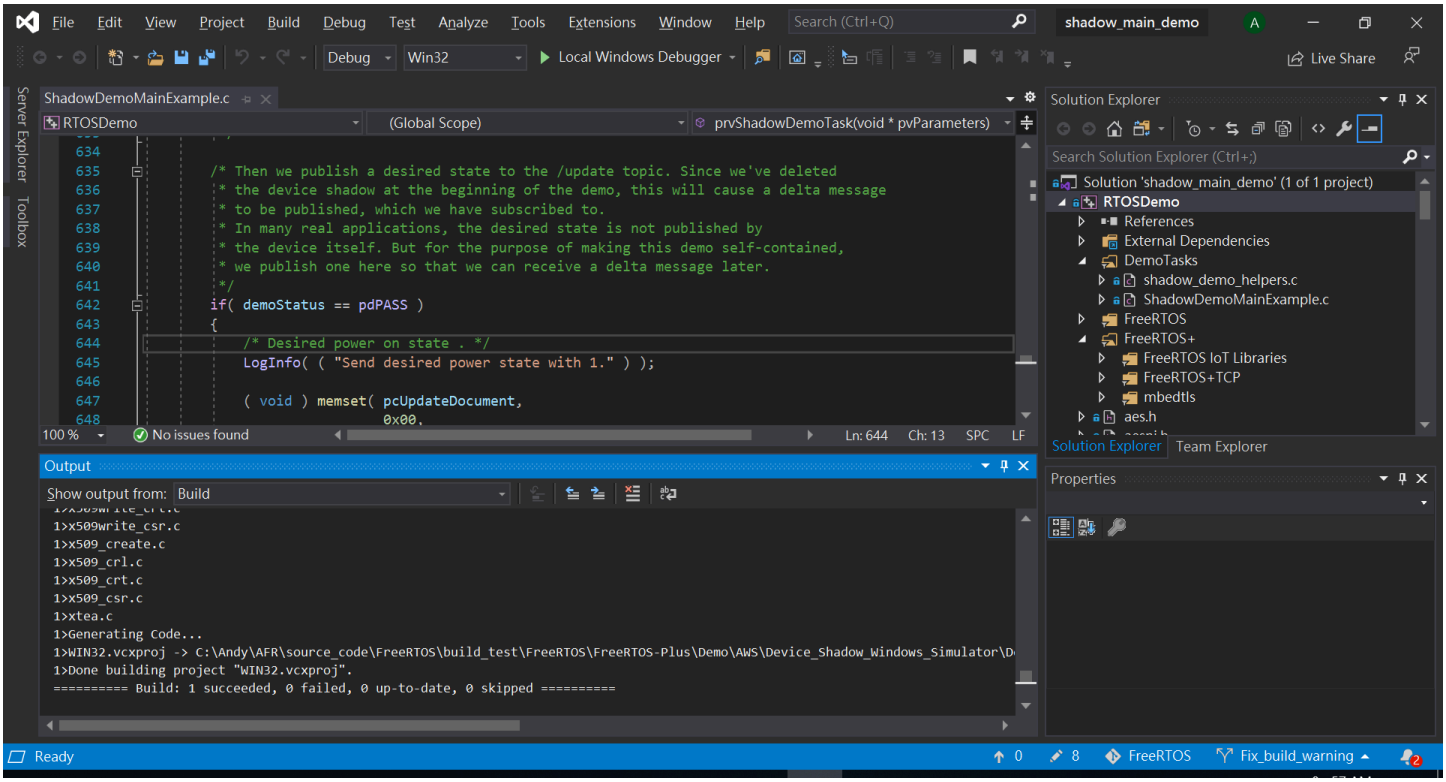
1. Estabelece uma conexão MQTT usando as funções auxiliares em `shadow_demo_helpers.c`.
2. Monta strings de tópicos MQTT para operações de sombra do dispositivo, usando macros definidas pela biblioteca do AWS IoT Device Shadow.
3. Publica no tópico MQTT usado para excluir uma sombra do dispositivo para 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 `power0n` 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 e determine se a mensagem está relacionada à sombra do dispositivo usando uma função definida pela biblioteca AWS IoT Device Shadow

()Shadow_MatchTopic. prvEventCallback 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.



A demonstração pode ser encontrada no arquivo *freertos*/demos/device_shadow_for_aws/shadow_demo_main.c ou em GitHub.

A captura de tela a seguir mostra a saída esperada quando a demonstração é bem-sucedida.



Conecte-se ao corretor AWS IoT MQTT

Para nos conectarmos ao corretor AWS IoT MQTT, usamos o mesmo método `MQTT_Connect()` doDemonstração de autenticação mútua da coreMQTT.

Exclusão de documento de sombra

Para excluir o documento paralelo, chame `xPublishToTopic` com uma mensagem vazia, usando macros definidas pela biblioteca 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 AWS IoT corretor 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 ativada prvEventCallback. 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 paralela com tipo`ShadowMessageTypeUpdateDelta`, chamaremos o 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 paralela com tipo`ShadowMessageTypeUpdateAccepted`, chamaremos o 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 pela aplicação. Se não corresponder, a função registrará uma mensagem de aviso.

Demonstração do cliente Echo de Secure Sockets

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 baixar FreeRTOS em. 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.
> Atualmente, as demonstrações do servidor TCP e do cliente não são suportadas nos kits de desenvolvimento Cypress CYW9439 07 AEVAL1 F e CYW9549 07 AEVAL1 F.

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_MQTT_DEMO_ENABLED` e defina `CONFIG_TCP_ECHO_CLIENT_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`.

Usando AWS IoT Device Tester para FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Você pode usar AWS IoT Device Tester (IDT) para FreeRTOS para verificar se o sistema operacional FreeRTOS funciona localmente em seu dispositivo e pode se comunicar com a nuvem. 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.

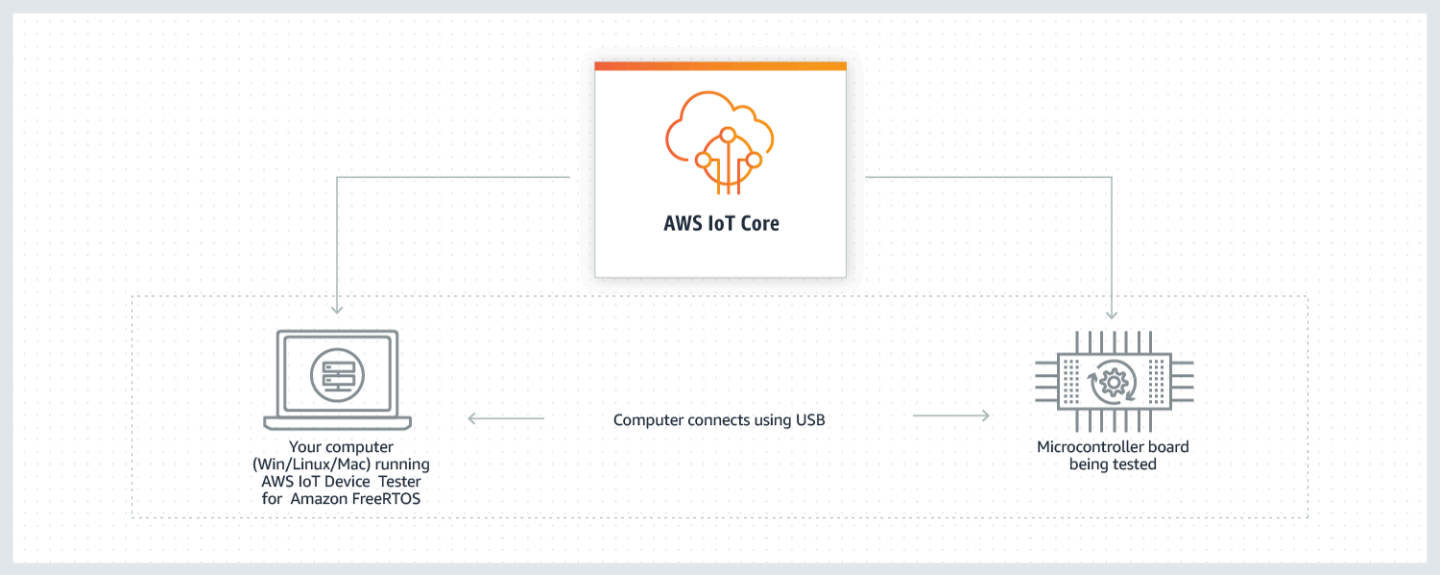
Além de testar dispositivos, o IDT para FreeRTOS cria recursos (por exemplo, coisas do AWS IoT , grupos de FreeRTOS, funções do Lambda e assim por diante) para facilitar o processo de qualificação.

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.

O diagrama a seguir mostra a configuração da infraestrutura de teste.



Para executar o IDT para FreeRTOS, você pode usar os recursos de teste. Existem dois tipos de recursos:

- Um conjunto 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.

Para obter mais informações, consulte AWS IoT Device Tester para versões do pacote de testes do FreeRTOS.

Versões suportadas do AWS IoT Device Tester para FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Este tópico lista as versões compatíveis do IDT 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. Novas versões do FreeRTOS podem exigir que você faça o download de uma nova versão do IDT para FreeRTOS.

Ao fazer download do software, você concorda com o Contrato de licença do IDT para FreeRTOS.

Versão mais recente do AWS IoT Device Tester for FreeRTOS

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

IDT v4.0.3 e conjunto de testes versão 1.5.0 para FreeRTOS 202012.00 (usam bibliotecas LTS do FreeRTOS 202012.00)

- IDT v4.0.3 com conjunto de testes FRQ_1.5.0 para Linux
- IDT v4.0.3 com conjunto de testes FRQ_1.5.0 para macOS
- IDT v4.0.3 com conjunto de testes FRQ_1.5.0 para Windows

> **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.

Notas de release

- Oferece suporte para qualificação de dispositivos com credenciais bloqueadas em um módulo de segurança de hardware.
- Correções de bugs secundários e melhorias.

Versões do pacote de testes

- FRQ_1.5.0
  - Lançado em 16/04/2021.

Versões anteriores do IDT para o FreeRTOS

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

IDT v4.0.1 e conjunto de testes versão 1.4.1 para FreeRTOS 202012.00 (usam bibliotecas LTS do FreeRTOS 202012.00)

- IDT v4.0.1 com conjunto de testes FRQ_1.4.1 para Linux
- IDT v4.0.1 com conjunto de testes FRQ_1.4.1 para macOS
- IDT v4.0.1 com conjunto de testes FRQ_1.4.1 para Windows

Notas da versão

- Oferece suporte ao FreeRTOS 202012.00 que usa bibliotecas LTS do FreeRTOS. Para obter mais informações sobre o que está incluído na versão 202012.00 do FreeRTOS, consulte o arquivo CHANGELOG.md em. GitHub
- Apresenta casos de teste OTA (Over-the-air) E2E (end-to-end) adicionais.
- Oferece suporte à qualificação de placas de desenvolvimento que executam o FreeRTOS 202012.00 e usam bibliotecas LTS do FreeRTOS.
- 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 suas próprias suítes de testes personalizadas usando AWS IoT Device Tester FreeRTOS. Para obter mais informações, consulte Usar o IDT para desenvolver e executar os próprios conjuntos de testes.

- Fornece aplicações IDT assinadas por código, para que não seja necessário conceder permissões ao executá-las no Windows ou no macOS.
- Refinou a lógica de análise dos resultados do teste de BLE.

Versões do pacote de testes

- FRQ_1.4.1
  - Lançado em 19/01/2021.

IDT v3.4.0 e conjunto de testes versão 1.3.0 para FreeRTOS 202011.01

- IDT v3.4.0 com conjunto de testes FRQ_1.3.0 para Linux
- IDT v3.4.0 com conjunto de testes FRQ_1.3.0 para macOS
- IDT v3.4.0 com conjunto de testes FRQ_1.3.0 para Windows

Notas da versão

- Oferece suporte ao FreeRTOS 202011.01. Para obter mais informações sobre o que está incluído na versão 202011.01 do FreeRTOS, consulte o arquivo CHANGELOG.md em. GitHub
- Corrigido o erro em que 'RSA' não era uma opção de PKCS11 configuração válida.
- Foi corrigido 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.

Versões do pacote de testes

- FRQ_1.3.0
  - Lançado em 05/11/2020.

IDT v3.3.0 e conjunto de testes versão 1.2.0 para FreeRTOS 202007.00

- IDT v3.3.0 com conjunto de testes FRQ_1.2.0 para Linux
- IDT v3.3.0 com conjunto de testes FRQ_1.2.0 para macOS
- IDT v3.3.0 com conjunto de testes FRQ_1.2.0 para Windows

Notas da versão

- Oferece suporte ao FreeRTOS 202007.00. Para obter mais informações sobre o que está incluído na versão do FreeRTOS 202007.00, consulte o arquivo CHANGELOG.md em. GitHub
- Novos testes de ponta a ponta para validar a suspensão de atualização sem fios e o recurso de retomada.
- Foi corrigido o erro que fazia com que os usuários na região eu-central-1 não conseguissem passar pela validação de configuração para testes OTA.
- Adicionado o parâmetro `--update-idt` ao comando `run-suite`. Você pode usar essa opção para definir a resposta para o prompt de atualização do IDT.
- Adicionado 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 atualização do arquivo de configuração para atualizações automáticas de pacote de testes.

Versões do pacote de testes

- FRQ_1.2.0
  - Lançado em 17/09/2020.
  - Melhorias internas e correções de erros, incluindo:
    - No OTA, agora o teste de desconexão e retomada usa corretamente o plano de dados de MQTT se ele estiver configurado.
    - Agora você pode alterar a senha da imagem do Raspberry Pi no BLE.

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

Versões não compatíveis do IDT para FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do 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 AWS IoT Device Tester para FreeRTOS.

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

IDT v3.0.2 para FreerTOS 202002.00

Notas da versão

- Oferece suporte ao FreeRTOS 202002.00. Para obter mais informações sobre o que está incluído na versão 202002.00 do FreeRTOS, consulte o arquivo CHANGELOG.md em. GitHub
- Adiciona atualização automática de conjuntos de testes no IDT. O IDT agora pode fazer download dos conjuntos 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 comando `upgrade-test-suite`.
  - Fazer download dos conjuntos de testes mais recentes definindo um sinalizador ao iniciar o IDT.

    Use a `-u` *`flag`* opção onde *`flag`* pode ser 'y' para sempre baixar ou 'n' para usar a versão existente.

    Quando houver várias versões do conjunto de testes 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 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 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.

Versões do pacote de testes

- FRQ_1.0.1

IDT v1.7.1 para FreerTOS 202002.00

Notas da versão

- Oferece suporte ao FreeRTOS 202002.00. Para obter mais informações sobre o que está incluído na versão 202002.00 do FreeRTOS, consulte o arquivo CHANGELOG.md em. GitHub
- Suporta o método de assinatura de código personalizado para casos de end-to-end teste over-the-air (OTA) para que você possa usar seus 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 testes falharão rapidamente com mensagens de erro melhoradas se a porta serial estiver mal configurada no arquivo `device.json`.
- Inclusão de 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 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`.

IDT v1.6.2 para FreeRTOS 201912.00

Notas da versão

- Oferece suporte ao FreeRTOS 201912.00.
- Oferece suporte a testes opcionais para OTA via HTTPS para qualificar suas placas de desenvolvimento FreeRTOS.
- Suporta o endpoint AWS IoT ATS em testes.
- Oferece suporte à capacidade de informar os usuários sobre a versão mais recente do IDT antes do início do conjunto de testes.

IDT v1.5.2 para FreeRTOS 201910.00

Notas da versão

- Oferece suporte à qualificação de dispositivos FreeRTOS com elemento seguro (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 multiplicador de tempo limite para aumentar os tempos limite, o que é útil ao solucionar problemas de erros relacionados ao tempo limite.
- Adicionada correção de erro para análise de log.
- Oferece suporte ao endpoint iot ats em testes.

IDT v1.4.1 para FreeRTOS 201908.00

Notas da versão

- Foi adicionado suporte para novas atualizações de PKCS11 bibliotecas e casos de teste.
- 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.

IDT v1.3.2 para FreeRTOS 201906.00 Major

Notas da versão

- 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

- Versões do FreeRTOS:
  - FreeRTOS v1.4.9
  - FreeRTOS v1.4.8
- Notas de release:
  - Adicionado suporte para FreeRTOS v1.4.8 e v1.4.9.

- Adicionado suporte para testes de dispositivos FreeRTOS com o sistema de compilação CMAKE.

IDT-FreeRTOS v1.1

- Versões do FreeRTOS:
  - FreeRTOS v1.4.7

IDT-FreeRTOS v1.0

- Versões do FreeRTOS:
  - FreeRTOS v1.4.6
  - FreeRTOS v1.4.5
  - FreeRTOS v1.4.4
  - FreeRTOS v1.4.3
  - FreeRTOS v1.4.2

Pré-requisitos

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 suportadas 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\`.

Qualificação LTS (qualificação para FreeRTOS que usam 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 suportadas 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.

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 a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica ou mensagem de texto 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 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, você pode visualizar a atividade atual da sua conta e gerenciar sua conta acessando https://aws.amazon.com/e escolhendo 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 a 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. Habilita o Centro de Identidade do IAM.

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

2. No Centro de Identidade do IAM, conceda o acesso administrativo a 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 o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

  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 Criar um conjunto de permissões 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 Adicionar grupos no Guia do usuário do AWS IAM Identity Center .

AWS IoT Device Tester política gerenciada

A política gerenciada pelo `AWSIoTDeviceTesterForFreeRTOSFullAccess` contém as seguintes permissões para habilitar o testador de dispositivos para executar e coletar métricas:

- `iot-device-tester:SupportedVersion`

Concede permissão para obter a lista de versões do FreeRTOS e versões do conjunto 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 disponível para download.

- `iot-device-tester:CheckVersion`

  Concede permissão para verificar se uma combinação de produto, suíte de testes e AWS IoT Device Tester versões é compatível.

- `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.

(Opcional) Instale o AWS Command Line Interface

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

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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 | Both"
                    }
                ]
            },
            {
                "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 sua placa suporta uma TCP/IP pilha e se ela é suportada no chip (MCU) ou descarregada em outro módulo. TCP/IP é necessário para a 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 EC2 instâncias AWS t2.micro e isso poderá gerar custos adicionais para sua conta. Para obter mais informações, consulte os EC2 preços da Amazon.

`TLS`

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

`PKCS11`

Indica o algoritmo de criptografia de chave pública que a placa suporta. PKCS11 é necessário para a 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.

`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 for impresso junto com a chave pública do dispositivo ao executar o projeto FreeRTOS demo/test .

`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": [
           "/absolute-path-to/build-parallel.sh {{testData.sourcePath}}
{{enableTests}}"
       ]
   },
   "flashTool": {
       "name": "your-flash-tool-name",
       "version": "your-flash-tool-version",
       "command": [
           "/absolute-path-to/flash-parallel.sh {{testData.sourcePath}}
{{device.connectivity.serialPort}} {{buildImageName}}"
       ],
       "buildImageInfo" : {
           "testsImageName": "tests-image-name",
           "demosImageName": "demos-image-name"
       }
   },
   "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-
path/aws_tests/config_files/core_pkcs11_config.h"
            },
            {
                "configName": "pkcs11TestConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
path/aws_tests/config_files/iot_test_pkcs11_config.h"
            }
        ],
        "optional": [
            {
                "configName": "otaAgentTestsConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
path/aws_tests/config_files/aws_ota_agent_config.h"
            },
            {
                "configName": "otaAgentDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
path/aws_demos/config_files/aws_ota_agent_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}}`.

`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`.

`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 caminho do SDK devem ser substituídas pela variável `{{sdkPath}}` do IDT para FreeRTOS.

`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 em que o firmware OTA está localizado. Alguns dispositivos não usam esse campo, mas é necessário 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`

A localização do certificado de assinatura de código no dispositivo.

`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`

`untrustedSignerCertificateArn`

O ARN do certificado de assinatura de código carregado para o ACM.

`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.

`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 portabilidade fornecido pelo FreeRTOS.

`cmakeConfiguration`

CMake configuração [Opcional]

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

`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 a cadeia 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 cujos arquivos de configuração você moveu, 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 CMake teste, você deve fornecer o nome da placa, o nome do fornecedor e o `afrToolchainPath` ou`compilerName`. Você também pode fornecer `cmakeToolchainPath` se você tem um caminho personalizado para o CMake conjunto de ferramentas.

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.

Execução de testes de Bluetooth Low Energy

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário 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 obter mais informações, consulte Versões do pacote 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 como`Y`, 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

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

```
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 obter mais informações, consulte Versões do pacote 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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.

AWS IoT Device Tester para versões do pacote de testes do FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

O IDT para FreeRTOS organiza testes em conjuntos de testes e grupos de testes.

- Um conjunto 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 testes individuais relacionados a um recurso específico, como PKCS, MQTT ou OTA.

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 AWS IoT Device Tester para FreeRTOS.

Você pode fazer download dos conjuntos de testes usando o comando `upgrade-test-suite`. Ou você pode usar o parâmetro opcional `-upgrade-test-suite` *`flag`* ao iniciar o IDT, onde *`flag`* pode ser 'y' para sempre baixar a 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.

Noções básicas de resultados e logs

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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.

`FRQ_Report.xml`É um relatório em formato JUnit XML padrão. Você pode integrá-lo em CI/CD plataformas 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 atributos 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 houver falhas de teste ou erros nos recursos necessários, seu dispositivo atende aos requisitos técnicos para executar FreeRTOS e pode interoperar com os serviços. 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 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*.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 conjuntos de testes

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 conjuntos de testes são compostos por três tipos de arquivos:

- Arquivos de configuração JSON que fornecem ao IDT informações sobre como executar o conjunto de testes.
- 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. Criação de arquivos de configuração JSON para seu conjunto de testes.
2. Crie executáveis de casos de teste que contenham a lógica de teste para seu conjunto de testes.

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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 conjunto de testes de amostra está estruturado e confira vários exemplos de executáveis de casos de teste e arquivos JSON de configuração de teste.

Próxima etapa: Usar o IDT para executar o conjunto de testes de amostra que você criou.

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.

Caso de teste não é executado

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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Um conjunto de testes combina o seguinte:

- Executáveis de teste que contêm a lógica de teste
- Arquivos de configuração JSON que descrevem o conjunto de testes

Este tutorial mostra como usar o IDT para FreeRTOS para desenvolver um conjunto de testes do 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. Criação de arquivos de configuração JSON
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 nessa pasta:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Criação de arquivos de configuração JSON

O conjunto de testes deve conter os seguintes arquivos de configuração JSON necessários:

Arquivos JSON necessários

`suite.json`

Contém informações sobre o conjunto de testes. Consulte Configuração do 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.

Caso de teste não é executado

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.

Criação de arquivos de configuração do conjunto de testes do IDT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Esta seção descreve os formatos nos quais você cria arquivos de configuração JSON incluídos ao escrever um conjunto de testes personalizado.

Arquivos JSON necessários

`suite.json`

Contém informações sobre o conjunto de testes. Consulte Configuração do 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 JSON opcionais

`state_machine.json`

Define como os testes são executados quando o IDT executa o conjunto de testes. Consulte Configuração de state_machine.json.

`userdata_schema.json`

Define o esquema do arquivo `userdata.json` que os executores de teste podem incluir em sua 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 Configuração de 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
    ### state_machine.json
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configuração do 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 testes. O valor de `id` deve corresponder ao nome da pasta do grupo de teste na qual o arquivo `group.json` está localizado. 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 Criação de executáveis de casos 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. 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 Criação de executáveis de casos 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 você especificar a mesma variável de ambiente no arquivo `test.json` e no arquivo `suite.json`, o valor no arquivo `test.json` prevalecerá.

Configuração de state_machine.json

Uma máquina de estado é uma estrutura que controla o fluxo de execução do conjunto de testes. 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 exibe 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 a máquina de estado do IDT funciona, consulte Configuração da máquina de estado do IDT.

Configuração de 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 conjunto de testes, 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 da máquina de estado.

Configuração da máquina de estado do IDT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Uma máquina de estado é uma estrutura que controla o fluxo de execução do conjunto de testes. 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 estado. *`state-name`*objeto contém a definição de um estado válido mapeado para o. *`state-name`*

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 casos IDs 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 `RunTask` estado para especificar um ID de grupo de teste sem especificar um caso de teste individual IDs, 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 caso de teste IDs especificados não for válido, esse estado emitirá o erro de `RunTaskError` execução. 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, em seguida, 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 caso a expressão definida em `Choices.Expression` seja 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 grupos de teste IDs. Todos os testes dentro de cada grupo de teste especificado devem ser aprovados para que o recurso seja compatível.

`OneOfGroups`

Opcional. Uma matriz de grupos de teste IDs. 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 variedade de casos de teste IDs. 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 você especificou um caso de teste individual IDs, `ResultVar` defina como *`group-id_test-id`*`_passed`.
- Se você não especificou um caso de teste individual IDs, `ResultVar` defina como *`group-id`*`_passed`.

O estado `AddProductFeatures` verifica os resultados dos testes da seguinte maneira:

- Se você não especificou nenhum caso de teste IDs, o resultado de cada grupo de teste será determinado a partir do valor da *`group-id`*`_passed` variável no contexto da máquina de estado.
- Se você especificou o caso de teste IDs, o resultado de cada um dos testes será determinado a partir do valor da *`group-id_test-id`*`_passed` variável 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 um grupo de teste válido IDs 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 toda a suíte de testes, essa chave será criada e conterá a lista de grupos de testes específicos. IDs

`specificTestCases`

Se o executor de teste selecionar casos de teste específicos para execução em vez de todo o conjunto de testes, essa chave será criada e conterá a lista de casos de teste específicos. IDs

`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.

Você pode consultar o contexto usando a JSONPath notação. A sintaxe para JSONPath consultas em definições de estado é. `{{$.query}}` Você pode usar JSONPath consultas como cadeias de caracteres de espaço reservado em alguns estados. O IDT substitui as cadeias de caracteres de espaço reservado pelo valor da JSONPath consulta avaliada do 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 JSONPath da notação para acessar dados do contexto, consulteUse 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"
        }
    }
}
```

Criação de executáveis de casos de teste do IDT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Você pode criar e colocar executáveis de casos de teste em uma pasta do conjunto de testes das seguintes formas:

- Para conjuntos de testes que usam argumentos ou variáveis de ambiente dos arquivos `test.json` para determinar quais testes executar, você pode criar um caso de teste executável único para todo o conjunto de testes ou um executável de teste para cada grupo de testes no conjunto de testes.
- 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 conjunto de testes 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.

Usar o IDT Client SDK

O IDT Client SDKs permite simplificar a forma como você escreve a lógica de teste em seu executável de teste com comandos de API que você pode usar para interagir com o IDT e seus dispositivos em teste. Atualmente, o IDT fornece o seguinte: SDKs

- SDK do cliente IDT para Python
- SDK for Go SDK para Go

Eles SDKs estão localizados na *`<device-tester-extract-location>`*`/sdks` pasta. 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
- Interação do IDT
- Interação do host

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 obter 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 obter 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 JUnit XML, você poderá 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Quando o IDT executa um conjunto de testes, o conjunto de testes 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
    },
    "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`

Informações do arquivo `config.json`. O campo `config` também contém o seguinte campo adicional:

`config.timeoutMultiplier`

O multiplicador de um valor de tempo limite usado pelo conjunto de testes. Este valor é especificado pelo executor de teste da CLI do IDT. O valor padrão é 1.

`device`

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

Você pode consultar o contexto usando a JSONPath notação dos seus arquivos JSON e do seu executável de texto com e. `GetContextValue` `GetContextString` APIs A sintaxe JSONPath das cadeias de caracteres 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 no SDKs.

A tabela a seguir descreve os operadores em uma expressão típica de foobar: JSONPath

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 valor JSONPath ou em sua expressão for um valor de matriz, booleano ou objeto, esse será o único operador binário compatível que você poderá usar.
  - >= (Maior ou igual a)
  - > (Maior que)
  - =~ (Correspondência de expressão regular). Para usar esse operador em uma expressão de filtro, o valor JSONPath ou no lado esquerdo da expressão deve ser avaliado como uma string e o lado direito deve ser um valor padrão que siga a RE2sintaxe.

Você pode usar JSONPath consultas no formato {{*query*}} como cadeias de caracteres de espaço reservado `environmentVariables` nos campos `args` e nos `test.json` arquivos e nos `environmentVariables` campos 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 JSONPath espaços reservados para recuperar cadeias de caracteres únicas para definir como variáveis de ambiente.

Definir configurações para os executores de teste

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Para executar conjuntos de testes personalizados, os executores de teste devem definir suas configurações com base no conjunto de testes que desejam executar. As configurações são especificadas com base nos modelos de arquivo de configuração JSON 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>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
```

```
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Todos 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 workload. 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 esse nome de recurso e definem o valor do recurso para os níveis de QOS compatíveis com o dispositivo deles. 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.

`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.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>,
                    "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 workload. 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.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 obter 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 `AWS_SECRET_ACCESS_KEY` ambiente `AWS_ACCESS_KEY_ID` e para armazenar 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>
```

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 no*<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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Depois de definir a configuração necessária, o IDT poderá executar o conjunto de testes. 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 comando a seguir 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 conjunto de testes nem executa a máquina de estado; em vez disso, ele interage com o IDE para responder às solicitações feitas do conjunto de testes em execução no IDE e exibe 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 a máquina de estado e não gera arquivos de relatório. Para depurar o conjunto de testes, você deve usar seu IDE para fornecer algumas informações que o IDT normalmente obtém dos arquivos JSON de configuração. Forneça 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 obter mais informações, consulte Executar o IDT no modo de depuração.

Analise os resultados e logs dos testes do IDT

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Esta seção descreve o formato no qual o IDT gera logs de 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.

```
time="2000-01-02T03:04:05-07:00" level=info msg=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.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

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 está sendo usado pelo caso de teste atual no momento.

Para exibir um resumo do teste no console quando um IDT terminar de executar um teste, você deve incluir um [estado `Report`] na máquina de estado. O resumo do teste contém informações sobre o conjunto de testes, os resultados de teste de cada grupo em execução e os locais dos arquivos de logs e relatórios 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 *suite-name*_Result.xml relatório 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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 *<device-tester-extract-location>* pasta
- 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 arquivo `config.jso` n para acessá-las.

Configure suas AWS credenciais

Se você ainda não tiver uma AWS conta, deverá criar uma. Se você já tem uma AWS conta, basta configurar as permissões necessárias para sua conta, permitindo que a IDT envie métricas de uso AWS em seu nome.

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 a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica ou mensagem de texto 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 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, você pode visualizar a atividade atual da sua conta e gerenciar sua conta acessando https://aws.amazon.com/e escolhendo 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 a 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. Habilita o Centro de Identidade do IAM.

   Para obter instruções, consulte Habilitar o AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .
2. No Centro de Identidade do IAM, conceda o acesso administrativo a 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 o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

  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 Criar um conjunto de permissões 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 Adicionar grupos no Guia do usuário do AWS IAM Identity Center .

Configurar permissões para o IDT

Nesta etapa, configure as permissões que o IDT usa para executar testes e coletar dados de uso do IDT. Você pode usar o AWS Management Console or AWS Command Line Interface (AWS CLI) para criar uma política do IAM e um usuário para o IDT e, em seguida, anexar políticas ao usuário.

- Para configurar permissões para IDT (Console)
- Para configurar permissões para o IDT (AWS CLI)

Como configurar permissões para o IDT (console)

Siga estas etapas para usar o console e configurar permissões para o IDT para FreeRTOS.

1. Faça login no console do IAM.
2. Crie uma política gerenciada pelo cliente que conceda permissões para criar funções com permissões específicas.

   a. No painel de navegação, selecione Políticas e, em seguida, Criar política.

   b. Na guia JSON, substitua o conteúdo do espaço reservado pela política a seguir.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

c. Escolha Review policy (Revisar política).

d. Em Name (Nome), insira **`IDTUsageMetricsIAMPermissions`**. Em Summary (Resumo), revise as permissões concedidas pela política.

e. Escolha Create policy (Criar política).

3. Crie um usuário do IAM e anexe permissões ao usuário.

   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.

Como configurar permissões para o IDT (AWS CLI)

Siga estas etapas para usar o AWS CLI para configurar as permissões do IDT para Freertos.

1. No seu computador, instale e configure o, AWS CLI se ainda não estiver instalado. Siga as etapas em Installing the (Instalando a) AWS CLI no Guia do usuário do AWS Command Line Interface .

> **Note**
>
> AWS CLI É uma ferramenta de código aberto que você pode usar para interagir com AWS serviços do seu shell de linha de comando.

2. Crie uma política gerenciada pelo cliente que conceda permissões para gerenciar o IDT e funções do FreeRTOS.

   Linux, macOS, or Unix

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "iot-device-tester:SendMetrics"
               ],
               "Resource": "*"
           }
       ]
   }'
   ```

   Windows command prompt

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document
                                           '{\"Version\": \"2012-10-17\",
    \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-
   tester:SendMetrics\"], \"Resource\": \"*\"}]}'
   ```

   > **Note**
   >
   > Esta etapa inclui um exemplo de prompt de comando do Windows porque ele usa uma sintaxe JSON diferente dos comandos de terminal Linux, macOS ou Unix.

3. Crie um usuário do IAM e anexe as permissões exigidas pelo IDT para FreeRTOS.

   a. Criar um usuário do IAM.

```
aws iam create-user --user-name user-name
```

b. Anexe a política de `IDTUsageMetricsIAMPermissions` criada para o novo usuário do IAM. *`user-name`*Substitua pelo nome de usuário do IAM e, *`<account-id>`* no comando, pelo ID da sua AWS conta.

```
aws iam attach-user-policy --user-name user-name --policy-arn
 arn:aws:iam::<account-id>:policy/IDTFreeRTOSIAMPermissions
```

4. Crie uma chave de acesso secreta para o usuário.

```
aws iam create-access-key --user-name user-name
```

Armazene a saída em um local seguro. Você usa essas informações posteriormente para configurar seu arquivo de AWS credenciais.

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 obter mais informações, consulte (Opcional) Configurar config.json.

Solução de problemas

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 você ver o erro "não *`user/role`* está autorizado a acessar este recurso", certifique-se de configurar as permissões conforme especificado em???.
2. Leia a saída do console para encontrar informações, como UUID de execução e tarefas atualmente em execução.
3. Examine o arquivo para verificar se há declarações de erro em cada teste `FRQ_Report.xml`. 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.

- Verifique se você tem uma versão limpa e clonada do código-fonte do FreeRTOS. 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 nesta documentação `execution-id`.

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 para cada caso de teste `/results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log`.

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	Certifique-se de que os campos obrigatór ios não estejam

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
		`userdata.json` estão ausentes ou em um formato incorreto.	ausentes e estejam no formato obrigatório nos arquivos listados. Para obter mais informações, consulte Preparação para testar sua placa de microcontrolador pela primeira vez.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
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: • AWS Região inválida - especifiq ue uma AWS região válida em seu `config.json` arquivo. Para obter mais informações sobre AWS regiões, consulte Regiões e endpoints. • AWS Credenciais inválidas - Defina AWS credencia is válidas em sua máquina de teste (por meio de variáveis de ambiente ou do arquivo de credenciais). Verifique se o campo de autentica ção está configura do corretame nte. Para obter mais informações, consulte ???.

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.

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, especifique `{{enableTests}}` 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:\Users\My`

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			`Name\Desktop \`, talvez sejam necessárias aspas adicionais dentro de seus comandos de compilaçã o para garantir que os caminhos sejam analisado s 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, atualizaç ão 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 de eco para os testes de soquetes seguros ou de soquetes. WiFi	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

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 realizar a validação manualmente em seu IDE, em um editor de texto como Atom ou Sublime ou por meio de uma ferramenta on-line como. JSONLint

Depuração de falhas na verificação de integridade

Ao executar o grupo de RTOSIntegrity teste Free e encontrar falhas, primeiro certifique-se de não ter modificado nenhum dos arquivos do *`freertos`* diretório. 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`*.

Depurando falhas do grupo de teste FullWiFi

Se você observar falhas no grupo de FullWiFi teste e o teste "`AFQP_WiFiConnectMultipleAP`" falhar, isso pode ser porque os dois pontos de acesso não estão na mesma sub-rede do computador host que executa 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 recursos 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.

Depuração de erro de "não é 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.

Depuração de erro de "não autorizado para acessar um recurso"

Você pode ver o erro "não *`user/role`* está autorizado a acessar este recurso" na saída do terminal ou no `test_manager.log` arquivo abaixo`/results/`*`execution-id`*`/logs`. Para resolver este problema, anexe a política gerenciada `AWSIoTDeviceTesterForFreeRTOSFullAccess` ao usuário de teste. Para obter mais informações, consulte ???.

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 enfrentando erros devido a tempos limite ou conexões indisponíveis nos testes de soquetes seguros WiFi ou de soquetes seguros, certifique-se de que sua rede esteja configurada para permitir tráfego para portas configuradas na faixa de 1024 a 49151.

O teste de soquetes seguros usa as portas 33333 e 33334 por padrão. Os WiFi testes 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 WRover dispositivo Espressif 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 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 AWS IoT Device Tester é executado, as falhas são reportadas ao 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. Em seguida, use essas informações para localizar e abrir o arquivo de log desse caso de teste chamado `results/execution-id/logs/test_group_id__test_case_id.log` As informações nesse arquivo incluem a saída completa dos comandos de compilação e flash, a saída da execução do teste e a saída mais detalhada AWS IoT Device Tester do console.

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
```

Recurso e AWS cobranças do celular

Quando o `Cellular` recurso estiver configurado `Yes` em seu `device.JSON` arquivo, FullSecureSockets usará EC2 instâncias t.micro para executar testes e isso poderá gerar custos adicionais em sua AWS conta. Para obter mais informações, consulte os EC2 preços da Amazon.

Política de geração de relatórios de qualificação

Os relatórios de qualificação são gerados somente pelas versões 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 de suporte AWS IoT Device Tester para FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

AWS IoT Device Tester for FreeRTOS é uma ferramenta de automação de testes para validar e qualificar seus dispositivos FreeRTOS para inclusão no Partner Device Catalog.AWS Recomendamos que você use a versão mais recente do FreeRTOS AWS IoT e do Device Tester para testar ou qualificar seus dispositivos. Oferecemos suporte ao AWS IoT Device Tester para a versão mais recente do FreeRTOS e para as versões do FreeRTOS lançadas nos seis meses anteriores. Uma versão AWS IoT Device Tester que suporta versões baseadas em LTS do FreeRTOS é suportada por dois anos. Atualmente, somente a versão AWS IoT Device Tester que suporta FreeRTOS 202012.00 é suportada por dois anos. A versão mais recente do FreeRTOS está disponível em. GitHub Para ver as versões compatíveis do AWS IoT Device Tester, consulteVersões suportadas do AWS IoT Device Tester para FreeRTOS.

Para cada versão da estrutura do IDT, três versões do conjunto de testes serão suportadas para qualificação de dispositivos.

Você também pode usar qualquer uma das versões compatíveis AWS IoT Device Tester com a versão correspondente dos FreeRTOS para testar ou qualificar seus dispositivos. Embora você possa continuar a usar Versões não compatíveis do IDT para FreeRTOS, elas não receberão correções de bugs ou atualizações.

Se você tiver dúvidas sobre a política de suporte, entre em contato com o Suporte ao cliente da AWS.

Segurança em AWS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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. Também existe a responsabilidade por outros fatores, incluindo a confidencialidade de dados, os requisitos da organização e as 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 para o FreeRTOS

- Validação de conformidade
- Resiliência em AWS
- Segurança da infraestrutura no FreeRTOS

Identity and Access Management para o FreeRTOS

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (fazer login) e autorizado (ter permissões) para usar recursos do FreeRTOS. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Tópicos

- Público
- Autenticação com identidades
- Gerenciar o acesso usando políticas
- Como o FreeRTOS funciona com o IAM
- Exemplos de políticas baseadas em identidade para o FreeRTOS
- Solução de problemas de identidade e acesso do FreeRTOS

Público

A forma como você usa AWS Identity and Access Management (IAM) difere, dependendo do trabalho que você faz nos FreeRTOS.

Usuário do serviço: se você usa o serviço FreeRTOS para fazer o trabalho, o administrador fornece as credenciais e as permissões necessárias. À medida que usar mais recursos do FreeRTOS para fazer seu trabalho, você poderá precisar de permissões adicionais. Compreenda como o acesso é gerenciado pode ajudar a solicitar as permissões corretas ao administrador. Se não for possível acessar um recurso no FreeRTOS, consulte Solução de problemas de identidade e acesso do FreeRTOS.

Administrador do serviço: se você for o responsável pelos recursos do FreeRTOS na empresa, provavelmente terá acesso total ao FreeRTOS. Cabe a você determinar quais funcionalidades

e recursos do FreeRTOS os usuários do serviço devem acessar. Envie as solicitações ao administrador do IAM para alterar as permissões dos usuários de serviço. Revise as informações nesta página para compreender os conceitos básicos do IAM. Para saber mais sobre como a empresa pode usar o IAM com o FreeRTOS, consulte Como o FreeRTOS funciona com o IAM.

Administrador do IAM: se você for um administrador do IAM, é aconselhável saber detalhes sobre como é possível criar políticas para gerenciar o acesso ao FreeRTOS. Para visualizar exemplos de políticas baseadas em identidade do FreeRTOS que podem ser usadas no IAM, consulte Exemplos de políticas baseadas em identidade para o FreeRTOS.

Autenticação com identidades

A autenticação é a forma como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado (conectado AWS) como o Usuário raiz da conta da AWS, como usuário do IAM ou assumindo uma função do IAM.

Você pode entrar AWS como uma identidade federada usando credenciais fornecidas por meio de uma fonte de identidade. AWS IAM Identity Center Usuários (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 faz login como identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. 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 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 designar solicitações por conta própria, consulte Versão 4 do AWS Signature para solicitações de API no Guia do usuário do IAM.

Independente do método de autenticação usado, também pode ser necessário fornecer 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 do usuário do AWS IAM Identity Center e Usar a autenticação multifator da AWS no IAM no Guia do usuário do IAM.

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 obter a lista completa das tarefas que exigem login como usuário-raiz, consulte Tarefas que exigem credenciais de usuário-raiz no Guia do Usuário do IAM.

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, é recomendável 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 mais informações sobre o Centro de Identidade do IAM, consulte O que é o Centro de Identidade do IAM? no Guia do Usuário do AWS IAM Identity Center .

Usuários e grupos do IAM

Um usuário do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas para uma única pessoa ou aplicativo. Sempre que possível, é recomendável contar com credenciais temporárias em vez de criar usuários do IAM com 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 usuários do IAM, é recomendável alternar as chaves de acesso. Para obter mais informações, consulte Alternar as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo no Guia do Usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. 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 recursos do IAM.

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 Casos de uso para usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Uma função do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. Para assumir temporariamente uma função do IAM no AWS Management Console, você pode alternar de um usuário para uma função do IAM (console). Você pode assumir uma função chamando uma operação de AWS API AWS CLI ou usando uma URL personalizada. Para obter mais informações sobre métodos para usar perfis, consulte Métodos para assumir um perfil no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, é possível 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 por ele. Para ter mais informações sobre perfis para federação, consulte Criar um perfil para um provedor de identidade de terceiros (federação) no Guia do usuário do IAM. Se usar o Centro de Identidade do IAM, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o Centro de Identidade do IAM correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de Permissões no Guia do Usuário do AWS IAM Identity Center .
- Permissões temporárias para usuários do IAM: um usuário ou um perfil do IAM pode presumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas: é possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta 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 conhecer a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

- 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 da entidade principal da chamada, usando um perfil de serviço ou um perfil vinculado ao serviço.
  - Sessões de acesso direto (FAS) — Quando você usa um usuário ou uma função do IAM para realizar ações AWS, você é considerado principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O FAS usa as permissões do diretor chamando um AWS service (Serviço da AWS), combinadas com a solicitação AWS service (Serviço da AWS) para fazer solicitações aos serviços posteriores. As solicitações do FAS 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 solicitações de FAS, consulte Sessões de acesso direto.
  - Perfil de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.
  - Função vinculada ao serviço — Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode presumir o perfil para executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para perfis vinculados a serviço.
- Aplicativos em execução na Amazon EC2 — Você pode usar uma função do IAM para gerenciar credenciais temporárias para aplicativos que estão sendo executados em uma EC2 instância e fazendo solicitações AWS CLI de AWS API. 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 Usar uma função do IAM para conceder permissões a aplicativos executados em EC2 instâncias da Amazon no Guia do usuário do IAM.

Gerenciar o 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 documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Os administradores podem usar políticas AWS JSON 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 perfis não têm permissões. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

As políticas do IAM definem permissões para uma ação independentemente do método usado para executar 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 da AWS Management Console AWS CLI, da ou da AWS API.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. 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 Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas 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 Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo 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 do IAM em uma política baseada 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. ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

O Amazon S3 e o AWS WAF Amazon VPC são exemplos de serviços que oferecem suporte. ACLs Para saber mais ACLs, 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 o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). 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 identidades do IAM no Guia do usuário do IAM.

- Políticas de controle de serviço (SCPs) — SCPs são políticas JSON 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. O SCP limita as permissões para entidades nas contas dos membros, incluindo cada uma Usuário raiz da conta da AWS. Para obter mais informações sobre Organizations e SCPs, consulte Políticas de controle de serviços no Guia AWS Organizations do Usuário.
- Políticas de controle de recursos (RCPs) — RCPs são políticas JSON que você pode usar para definir o máximo de permissões disponíveis para recursos em suas contas sem atualizar as políticas do IAM anexadas a cada recurso que você possui. O RCP limita as permissões para recursos nas contas dos membros e pode afetar as permissões efetivas para identidades, incluindo a Usuário raiz da conta da AWS, independentemente de pertencerem à sua organização. Para obter mais informações sobre Organizations e RCPs, incluindo uma lista Serviços da AWS desse suporte RCPs, consulte Políticas de controle de recursos (RCPs) 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 recursos. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte Políticas de sessão no Guia do usuário do IAM.

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 políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

Como o FreeRTOS funciona com o IAM

Antes de usar o IAM para gerenciar o acesso ao FreeRTOS, saiba quais recursos do IAM estão disponíveis para usar com o FreeRTOS.

recursos do IAM que você pode usar com o FreeRTOS

Atributo do IAM	Suporte ao FreeRTOS
Políticas baseadas em identidade	Sim
Políticas baseadas em recurso	Não
Ações de políticas	Sim
Recursos de políticas	Sim
Chaves de condição de política (específicas do serviço)	Sim
ACLs	Não
ABAC (tags em políticas)	Parcial
Credenciais temporárias	Sim
Permissões de entidade principal	Sim
Perfis de serviço	Sim
Perfis vinculados a serviço	Não

Para ter uma visão de alto nível de como os FreeRTOS e AWS outros serviços funcionam com a maioria dos recursos do IAM, AWS consulte os serviços que funcionam com o IAM no Guia do usuário do IAM.

Políticas baseadas em identidade para o FreeRTOS

Compatível com políticas baseadas em identidade: sim

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário do IAM, grupo de usuários ou perfil. 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 Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

Com as políticas baseadas em identidade do IAM, é possível especificar ações e recursos permitidos ou negados, assim 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 podem ser usados em uma política JSON, consulte Referência de elemento de política JSON do IAM no Guia do usuário do IAM.

Exemplos de políticas baseadas em identidade para o FreeRTOS

Para visualizar exemplos de políticas baseadas em identidade do FreeRTOS, consulte Exemplos de políticas baseadas em identidade para o FreeRTOS.

Políticas baseadas em recursos no FreeRTOS

Compatibilidade com políticas baseadas em recursos: não

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo 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 permitir o acesso entre contas, você pode especificar uma conta inteira ou as entidades do IAM em outra conta como a entidade 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 administrador do IAM 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. Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Ações de políticas para FreeRTOS

Compatível com ações de políticas: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. As ações de política geralmente têm o mesmo nome da operação de AWS API associada. Existem algumas exceções, como ações somente de permissão, que não têm uma operação de API correspondente. 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 visualizar uma lista das ações do FreeRTOS, consulte Ações definidas pelo FreeRTOS na Referência de autorização do serviço.

As ações de políticas no FreeRTOS usam o seguinte prefixo antes da ação:

```
awes
```

Para especificar várias ações em uma única declaração, separe-as com vírgulas.

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

Para visualizar exemplos de políticas baseadas em identidade do FreeRTOS, consulte Exemplos de políticas baseadas em identidade para o FreeRTOS.

Recursos de políticas para o FreeRTOS

Compatível com recursos de políticas: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou `NotResource`. Como prática

recomendada, especifique um recurso usando seu nome do recurso da Amazon (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 recursos do FreeRTOS e ARNs seus, consulte Recursos definidos pelo FreeRTOS na Referência de Autorização de Serviço. Para saber com quais ações é possível especificar o ARN de cada recurso, consulte Ações definidas pelo FreeRTOS.

Para visualizar exemplos de políticas baseadas em identidade do FreeRTOS, consulte Exemplos de políticas baseadas em identidade para o FreeRTOS.

Chaves de condição de políticas para o FreeRTOS

Compatível com chaves de condição de política específicas de serviço: sim

Os administradores podem usar políticas AWS JSON 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 de `Condition` em uma declaração ou várias chaves em um único elemento de `Condition`, 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, é possível conceder a um usuário do IAM permissão para acessar um recurso somente se ele estiver marcado com seu nome de usuário do IAM. Para obter mais informações, consulte Elementos da política do IAM: variáveis e tags no Guia do usuário do IAM.

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 do usuário do IAM.

Para visualizar uma lista de chaves de condição do FreeRTOS, consulte Chaves de condição do FreeRTOS na Referência de autorização do serviço. Para saber com quais ações e recursos é possível usar uma chave de condição, consulte Ações definidas pelo FreeRTOS.

Para visualizar exemplos de políticas baseadas em identidade do FreeRTOS, consulte Exemplos de políticas baseadas em identidade para o FreeRTOS.

ACLs em FreeRTOS

Suportes ACLs: Não

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

ABAC com o FreeRTOS

Compatível com ABAC (tags em políticas): parcial

O controle de acesso por atributo (ABAC) é uma estratégia de autorização que define as permissões com base em atributos. Em AWS, esses atributos são chamados de tags. Você pode anexar tags a entidades do IAM (usuários ou funções) e a vários AWS recursos. Marcar de entidades e atributos é a primeira etapa do ABAC. Em seguida, você cria políticas de ABAC para permitir operações quando a tag da entidade principal corresponder à tag do recurso que ela estiver tentando acessar.

O ABAC é útil em ambientes que estão crescendo rapidamente e ajuda em situações em que o gerenciamento de políticas se torna um problema.

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 for compatível com as três chaves de condição para cada tipo de recurso, o valor será Sim para o serviço. Se um serviço for compatível com as três chaves de condição somente para alguns tipos de recursos, o valor será Parcial

Para obter mais informações sobre o ABAC, consulte Definir permissões com autorização do ABAC no Guia do usuário do IAM. Para visualizar um tutorial com etapas para configurar o ABAC, consulte Usar controle de acesso baseado em atributos (ABAC) no Guia do usuário do IAM.

Usar credenciais temporárias com o FreeRTOS

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 trabalhar com o IAM no Guia do usuário do IAM.

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 como alternar funções, consulte Alternar para um perfil do IAM (console) no Guia do usuário do IAM.

Você pode criar manualmente credenciais temporárias usando a AWS API AWS CLI ou. 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 no IAM.

Permissões de entidade principal entre serviços para o FreeRTOS

Compatibilidade com o recurso de encaminhamento de sessões de acesso (FAS): sim

Quando você usa um usuário ou uma função do IAM para realizar ações AWS, você é considerado um principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O FAS usa as permissões do diretor chamando um AWS service (Serviço da AWS), combinadas com a solicitação AWS service (Serviço da AWS) para fazer solicitações aos serviços posteriores. As solicitações do FAS 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 solicitações de FAS, consulte Sessões de acesso direto.

Perfis de serviço para o FreerTOS

Compatível com perfis de serviço: sim

O perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.

> **Warning**
>
> Alterar as permissões de um perfil de serviço pode prejudicar a funcionalidade do FreeRTOS. Apenas edite os perfis de serviço quando o FreeRTOS orientar você a fazer isso.

Perfis vinculados ao serviço para o FreeRTOS

Compatível com perfis vinculados ao serviço: Não

Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode presumir o perfil para executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para funções vinculadas ao serviço.

Para obter detalhes sobre como criar ou gerenciar perfis vinculados a serviços, consulte Serviços da AWS que funcionam com o IAM. Encontre um serviço na tabela que inclua um `Yes` na coluna Perfil vinculado ao serviço. Escolha o link Sim para visualizar a documentação do perfil vinculado a serviço desse serviço.

Exemplos de políticas baseadas em identidade para o FreeRTOS

Por padrão, usuários e perfis não têm permissão para criar ou modificar recursos do FreeRTOS. Eles também não podem realizar tarefas usando a AWS API AWS Management Console, AWS Command Line Interface (AWS CLI) ou. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

Para aprender a criar uma política baseada em identidade do IAM ao usar esses documentos de política em JSON de exemplo, consulte Criar políticas do IAM (console) no Guia do usuário do IAM.

Para obter detalhes sobre ações e tipos de recursos definidos pelo FreeRTOS, incluindo o formato do para cada um ARNs dos tipos de recursos, consulte Ações, recursos e chaves de condição do FreeRTOS na Referência de Autorização de Serviço.

Tópicos

- Práticas recomendadas de política
- Uso do console do FreeRTOS
- Permitir que os usuários visualizem suas próprias permissões

Práticas recomendadas de política

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir recursos do FreeRTOS em sua conta. Essas ações podem incorrer em custos para sua 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 gerenciadas pela AWS ou Políticas gerenciadas pela AWS para funções de trabalho no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre como usar o IAM para aplicar permissões, consulte Políticas e permissões no IAM no Guia do usuário do IAM.
- Use condições nas políticas do IAM para restringir ainda mais o acesso: você pode adicionar uma condição às 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 usando SSL. Você também pode usar condições para conceder acesso às ações de serviço se elas forem usadas por meio de uma ação específica AWS service (Serviço da AWS), como AWS CloudFormation. Para obter mais informações, consulte Elementos da política JSON do IAM: condição no Guia do usuário do IAM.
- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para obter mais informações, consulte Validação de políticas do IAM Access Analyzer no Guia do Usuário do IAM.

- Exigir autenticação multifator (MFA) — Se você tiver um cenário que exija usuários do IAM ou um usuário root, ative Conta da AWS a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para obter mais informações, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para obter mais informações sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Uso do console do FreeRTOS

Para acessar o console do FreeRTOS, você deve ter um conjunto mínimo de permissões. Essas permissões devem permitir que você liste e visualize detalhes sobre os recursos do FreeRTOS em seu. Conta da AWS Caso crie uma política baseada em identidade mais restritiva 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 a API AWS CLI ou para a AWS API. Em vez disso, permita o acesso somente a ações que correspondam à operação de API que estiverem tentando executar.

Para garantir que usuários e funções ainda possam usar o console do FreeRTOS, anexe também o FreeRTOS ou a política gerenciada às *ConsoleAccess* entidades *ReadOnly* AWS . Para obter informações, consulte Adicionar permissões a um usuário no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como criar uma política que permita que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou programaticamente usando a API AWS CLI ou AWS .

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
```

```
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Solução de problemas de identidade e acesso do FreeRTOS

Use as seguintes informações para ajudar a diagnosticar e corrigir problemas comuns que podem ser encontrados ao trabalhar com o FreeRTOS e o IAM.

Tópicos

- Não tenho autorização para executar uma ação no FreeRTOS
- Não estou autorizado a realizar iam: PassRole
- Quero permitir que pessoas de fora da minha acessem meus Conta da AWS recursos do FreeRTOS

Não tenho autorização para executar uma ação no FreeRTOS

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, suas políticas deverão ser atualizadas para permitir que você realize a ação.

O erro do exemplo a seguir ocorre quando o usuário do IAM `mateojackson` tenta usar o console para visualizar detalhes sobre um atributo *`my-example-widget`* fictício, mas não tem as permissões `awes:`*`GetWidget`* fictícias.

```
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 uma mensagem de erro informando que não tem autorização para executar a ação `iam:PassRole`, as suas políticas deverão ser atualizadas para permitir a passagem de um perfil 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 erro de exemplo a seguir ocorre quando uma usuária do IAM chamada`marymajor` tenta usar o console para executar 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 de fora da minha acessem meus Conta da AWS recursos do FreeRTOS

Você pode criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível 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 FreeRTOS oferece suporte a esses recursos, consulte Como o FreeRTOS funciona com o IAM.
- Para saber como fornecer acesso aos seus recursos em todos os Contas da AWS que você possui, consulte Como fornecer acesso a um usuário do IAM em outro Conta da AWS que você possui no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como fornecer acesso Contas da AWS a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Validação de conformidade

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

Para saber se um AWS service (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 regulamentos aplicáveis. AWS fornece os seguintes recursos para ajudar na conformidade:

- Governança e conformidade de segurança: esses guias de implementação de solução abordam considerações sobre a arquitetura e fornecem etapas para implantar recursos de segurança e conformidade.
- Referência de serviços qualificados para HIPAA: lista os serviços qualificados para HIPAA. Nem todos Serviços da AWS são elegíveis para a HIPAA.
- 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 AWS service (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 AWS service (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, como o PCI DSS, atendendo aos requisitos de detecção de intrusões exigidos por determinadas estruturas de conformidade.
- AWS Audit Manager— Isso AWS service (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

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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, é possível 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 data center tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte Infraestrutura AWS global.

Segurança da infraestrutura no FreeRTOS

> **Important**
>
> Esta é uma versão arquivada do Guia do usuário do FreeRTOS para usar com a versão 202012.00 do FreeRTOS. Para obter a versão mais recente deste documento, consulte o Guia do usuário do FreeRTOS.

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 chamadas de API AWS publicadas para acessar AWS serviços pela rede. Os clientes devem oferecer suporte a Transport Layer Security (TLS) 1.2 ou posterior. Recomendamos usar o TLS 1.3 ou posterior. Os clientes também devem ter compatibilidade com conjuntos de criptografia com perfect forward secrecy (PFS) como Ephemeral Diffie-Hellman (DHE) ou Ephemeral Elliptic Curve Diffie-Hellman (ECDHE). A maioria dos sistemas modernos como Java 7 e versões posteriores oferece compatibilidade com esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou você pode usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

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.